ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO DE TELECOMUNICACIÓN

Título del proyecto:

SISTEMA DE MONITORIZACIÓN INALÁMBRICO DE TEMPERATURA PARA DISPOSITIVOS ANDROID

Juniors Antonio Medina Landeón

Javier Goicoechea Fernández e Ignacio del Villar

Pamplona, 8 de septiembre 2016

Contenido

AGRADECIMIENTOS

Es imposible empezar a redactar este proyecto final de carrera sin antes mostrar mis agradecimientos, en primer lugar a Dios, ya que con él todo es posible y sin él todo está perdido, en segundo lugar a mi madre, Raida Fiorella Landeón Vicuña, que en resumidas cuentas es mi fuerza, mi inspiración y mi voluntad de seguir adelante, ella es la persona más importante que tengo en mi vida, sin ella nunca hubiese sido posible que yo estuviera en este hermoso país y por ende terminar la carrera en esta prestigiosa universidad, en tercer lugar a mi padre, Aquiles Antonio Medina de Paz, que desde la lejanía de mi Perú, siempre me alienta a seguir adelante, en cuarto lugar a mis tutores Javier Goicoechea e Ignacio Del Villar que personalmente los admiro por sus enormes conocimientos, son para mí unas fuentes de sabiduría sin lugar a duda, me han tenido una paciencia casi infinita, y cada vez que acudía a sus despachos para resolver dudas respecto a este trabajo, me han tratado de la mejor manera posible y me han sabido nutrir de conocimientos, en quinto lugar ami gran amigo Victor Hugo Alvarado Chahua, que lo conocí recién ni bien llegar a este hermoso país, y me abrió las puertas de su amistad y desde que comencé la universidad hasta el término de ésta, ha sabido apoyarme como nunca me imaginé, se ha ganado mis respetos, reconocimiento y un eterno agradecimiento, en último lugar y por ello no menos importante a mi gran amiga y también amiga de la familia, Aurelia Lacunza que fue junto con mi madre el motor causante de estar yo, hoy aquí presente desde hace 10 años en España, es una persona que vale su peso en oro, muy culta, buena con un altruismo estratosférico y cada conversación con ella es de mucho provecho, y siendo sincero, aprendo mucho de ella.

A todas estas personas y a otras más de las que no he hecho mención, gracias, gracias y millones de gracias, los quiero….

CAPÍTULO 1. INTRODUCCIÓN Y OBJETIVOS

En la actualidad la evolución de los dispositivos electrónicos ha hecho que casi todo el mundo lleve un ordenador en el bolsillo. La proliferación de microprocesadores de altas prestaciones y con un consumo de energía muy bajo ha hecho posible el desarrollo y comercialización masiva a precio competitivo de dispositivos como los *Smartphones* o las *Tablets*. Además el desarrollo de sistemas operativos de código abierto en el que los desarrolladores pueden programar sus propias aplicaciones y distribuirlas libremente hace que se abran muchas potenciales oportunidades para la creación de nuevos productos.

En este contexto se desea construir una pequeña cámara climática para el laboratorio de sensores de la Universidad Pública de Navarra y que este dispositivo se comunique de forma inalámbrica con dispositivos portátiles, ya sea *smartphones* o *tablets*. Para esto, el presente proyecto se centrará en la adquisición de datos de temperatura de un espacio, su transmisión inalámbrica y en el diseño y creación de una aplicación Android que permite el monitoreo constante de este parámetro. Para monitorizar la temperatura y controlarla mediante un algoritmo de control automático y un actuador electrotérmico se empleará la placa de desarrollo ARDUINO UNO (aunque también se dispone de la placa ARDUINO MEGA). Además esta placa electrónica servirá para enviar de forma inalámbrica en tiempo real los datos que se adquieran mediante una comunicación *Bluetooth*.

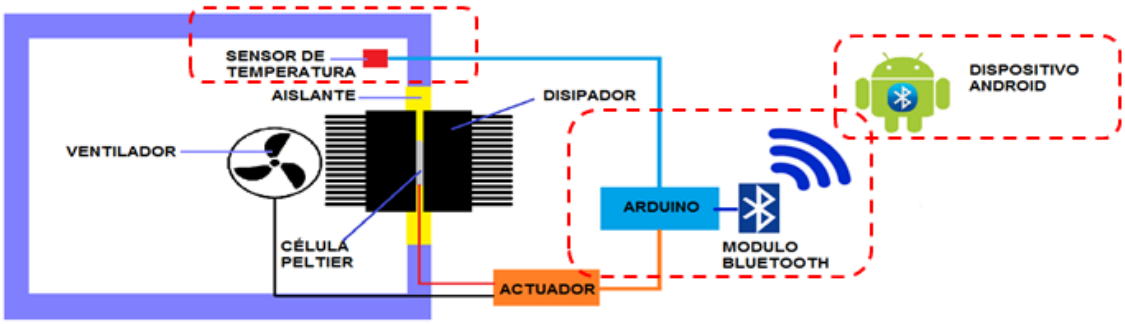


Figura 1-1 Esquema-objetivo del presente proyecto.

De esta manera, una vez adquiridos los datos de temperatura, en este proyecto se desarrollará una comunicación Bluetooth entre una aplicación Android y la placa de desarrollo Arduino, con el fin de registrar la temperatura interna de una cámara climática en un dispositivo Android.

Para crear la aplicación Android se utilizará la plataforma Android Studio en la que se programarán los elementos necesarios para establecer la comunicación, y también para la recepción y la visualización de los datos.

Cabe resaltar que se ha utilizado el sensor digital de temperatura SHT15, el cual conllevaba su respectiva programación en lenguaje Arduino.

CAPÍTULO 2. CONCEPTOS PREVIOS

Es necesario antes de entrar propiamente en materia de éste proyecto conocer y comprender ciertos conceptos que irán apareciendo a lo largo del mismo.

2.1. ¿QUÉ ES ARDUINO?

Sería muy fácil decir que Arduino, es una placa electrónica que contiene un microcontrolador y que su principal característica es la facilidad con la que se programa, a diferencia de las demás placas con microcontroladores del mercado que su programación es más laboriosa. Pero Arduino es más que eso, es en realidad tres cosas:

Una placa hardware (PCB, del inglés **"**printed circuit board", es decir placa de circuito impreso) que contiene un microcontrolador reprogramable y una serie de pines tipo hembra, que están internamente unidos a las patillas de entrada/salida (E/S) del microcontrolador, que permiten conectar de una forma muy sencilla y cómoda diferente distintos tipos de sensores y actuadores.

Figura 2-1 Placa Arduino MEGA.

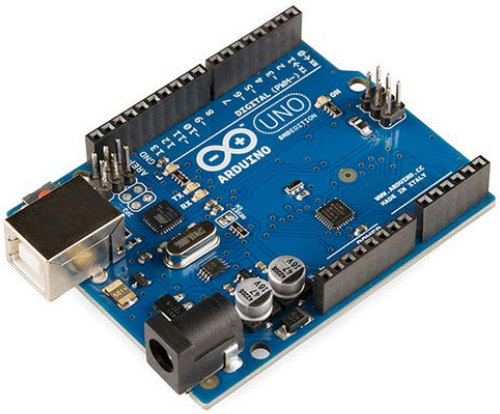


Figura 2-2 Placa Arduino UNO (el que se usará).

Un entorno de desarrollo, conocido como IDE del inglés "integrated development environment", el cual es un software gratis, libre y multiplataforma, ésta última característica se debe a que funciona en Linux, MacOs y Windows. Éste software que se debe instalar en nuestro ordenador nos permitirá escribir, verificar y cargar en la memoria del microcontrolador de la placa Arduino el conjunto de instrucciones (nuestro programa) que queremos que se ejecute.

Es decir gracias al IDE podemos programar el microcontrolador, y la manera en la que se conecta nuestro ordenador con la placa (para poder enviar las instrucciones de código y grabarle éstas) es por medio de un cable USB, ya que las placas Arduino incorporan un conector de éste tipo.

Los programas escritos en Arduino y cargados en el microcontrolador (proyectos Arduino) pueden ser autónomos energéticamente hablando o no. Es decir que pueden estar ligados al ordenador o no, en el primer caso, mediante un cable USB, cable de red Ethernet, etc., el ordenador hará de fuente de alimentación, en el segundo caso la autonomía viene dada por el simple hecho de que la alimentación viene dada por alguna fuente externa como puede ser una pila de 9 voltios.

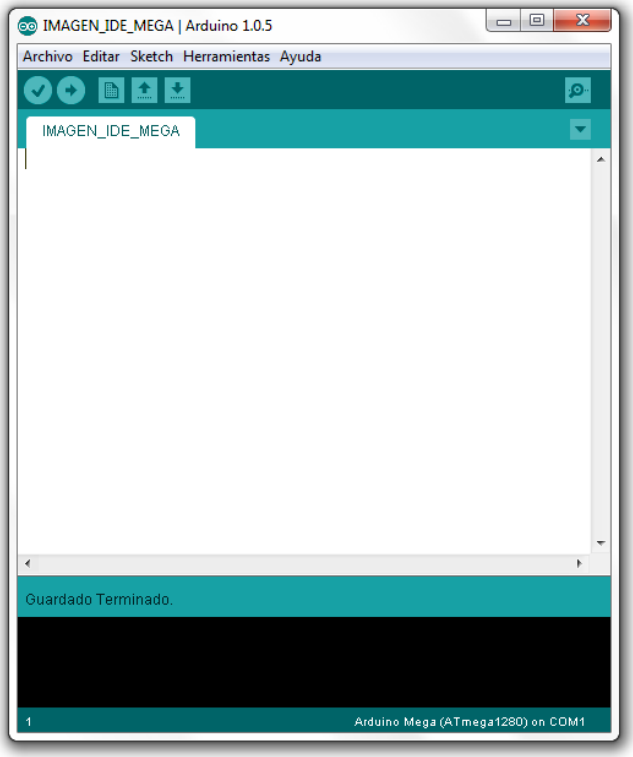


Figura 2-2 IDE de Arduino.

Un lenguaje de programación libre, es decir que no tenemos que pagar licencias, se entiende por lenguaje de programación, a aquel lenguaje que hace posible que el ordenador nos entienda, para que ejecute unas instrucciones que queremos que se lleve

a cabo. El lenguaje de programación Arduino, es un lenguaje sencillo que conserva los bloques condicionales (if y else/if), bloques repetitivos (while, do-while, for), las instrucciones break y continue, y demás elementos como variables, funciones, etc., que encontramos en muchos lenguajes de programación existentes.

2.2 ¿QUÉ ES ANDROID?

Android es un sistema operativo creado inicialmente para teléfonos móviles (extendido actualmente para relojes, tablets, ordenadores portátiles e incluso televisores) al igual que Apple IOS, Windows Phone (sucesor de Windows Mobile), BlackBerry, Symbian, entre otros.

Android está basado en el sistema operativo Linux, lo cual hace que Android sea libre, gratuito y multiplataforma.

Éste sistema operativo Android nos permite programar aplicaciones en Java, un lenguaje de programación orientado a objetos muy conocido, la denominación de que sea orientado a objeto, será explicada en los siguientes apartados. Como dijimos anteriormente una de las mejores características de Android es que es libre (completamente libre), es decir que para programar en Android ni para incluir nuestras aplicaciones no necesitamos pagar nada, es más podemos abrir el código fuente de cualquier aplicación Android, inspeccionarlo, detectar errores e incluso mejorarlo, es por esto y muchas cosas más que Android es una de las plataformas más demandadas en el mundo de la telefonía móvil, tanto para usuarios como para desarrolladores.

Figura 2-3 Logotipo de Android.

2.3 ¿QUÉ ES JAVA?

Java es un lenguaje de programación y una plataforma que lleva el mismo nombre (conocida como máquina virtual Java),respecto a la definición como lenguaje de programación, lo es, pero un lenguaje de programación orientado a objetos, ¿Y qué quiere decir esto? Que todo va a estar enfocado a objetos, los cuales vamos a utilizar para poder haces nuestros programas. Ciertamente hablar de la programación orientada a objetos, es hablar de una manera más fácil de entender la programación como tal, a groso modo es como si modelaríamos la realidad por medio de la programación, en el próximo capítulo se verá un poco más detenidamente en que consiste la programación orientada a objetos (POO), y a que hace referencia la palabra "objeto".

Java como plataforma o mejor dicho como máquina virtual es un software muy útil y posiblemente me quede corto llamándolo útil, ya que Java está en casi todos lados, es decir tiene aplicaciones en muchísimas cosas, como por ejemplo se pueden realizar aplicaciones de escritorio del ordenador, aplicaciones móviles, se pueden programar hasta firmware, es decir software que tienen los electrodomésticos (por poner un ejemplo), incluso para poder utilizar los programas avanzados y recientes del momento (que compramos o descargamos), hasta los reproductores de blu ray tienen código en java, obviamente la mayor utilización de java está en la web, hay muchas aplicaciones en internet, juegos en red, páginas o sitios de internet que están hechos en java, a día de hoy más de 13000 millones de dispositivos usan Java.

A continuación vamos a ver las características principales del lenguaje de programación Java:

SIMPLE: El lenguaje de programación java es simple, ya que su sintaxis es muy parecida a la de C++ pero simplificada, los creadores de java , se basaron en la sintaxis de C++, pero quitando de éste todo lo que resultase complicado y los fuentes de errores de éste (Java trata de hacer las cosas más simples).

INDEPENDIENTE DE LA PLATAFORMA: Java es independiente de la plataforma en la que estemos trabajando es decir, nosotros podemos realizar nuestros programas JAVA, y éstos se pueden ejecutar (coloquialmente "funcionar") en cualquier sistema operativo

que implemente una máquina virtual java, es decir por ejemplo Junior puede realizar un programa Java en Windows y ustedes pueden ejecutar ese programa en Windows, Linux, Mac, Solaris, etc., siempre que ese sistema operativo tenga la implementación de la máquina virtual Java (es por esa razón que Java es un lenguaje interpretado, el intérprete es la máquina virtual Java), la cual es un programa que tiene una arquitectura bien definida, dedicaré un subapartado entero para tratar el funcionamiento de la máquina virtual Java, ya que desde mi punto de vista es muy importante entenderla del todo.

LENGUAJE INTERPRETADO: Como dije antes, para ejecutar las aplicaciones hechas en Java, necesitamos el intérprete de Java que es la máquina virtual Java.

DISTRIBUIDO: Gracias a las clases que posee Java, vamos a poder desarrollar aplicaciones en un entorno de red, por medio de sockets (interfaces de red), siguiendo una serie de protocolos definidos, interconectando procesos remotos o alejados entre sí, es decir vamos a poder desarrollar aplicaciones distribuidas.

SEGURO: Java es especialmente seguro, soporta la seguridad "sandboxing" o "caja de arena", dado que la máquina virtual Java realmente es un software, ningún programa en java toma el control del procesador, esto nos permite aislar lo que es la máquina virtual del entorno real (es decir del sistema operativo y del hardware). A modo de curiosidad una "sandbox" es un lugar cerrado, **seguro** y lleno de arena para que los niños jueguen, y en términos de informática la "sandbox", es una zona de memoria, totalmente aislada del resto, donde se puede ejecutar cualquier tipo de software, especialmente de origen desconocido o "malintencionado" sin que se produzca infección alguna a nuestro sistema operativo o cualquier componente hardware, y esto es así ya que se le impide acceder a cualquier otra zona que no sea la "sandbox".

ROBUSTO: Java también es robusto ya que al ejecutarse los programas dentro de la máquina virtual Java (M.V.J) se impide que el sistema se bloquee, en java la asignación entre tipos distintos no es posible, a diferencia de C, por poner un ejemplo no se puede declarar un tipo int a un char. Otro aspecto a mencionar es la gestión de memoria, Java no dispone de punteros para acceder a una posición de memoria, es el sistema interno de Java que se encarga de gestionar y liberar memoria, facilitándonos la vida a todos los

programadores. Por último mencionar que el código es chequeado o verificado dos veces, primero cuando se compila y se genera los llamados “bytecodes”. Y finalmente hay un segundo chequeo en el momento en que llega la hora de interpretar los bytecodes por la máquina virtual Java.

MULTIHILOS: En los tiempos que estamos viviendo tanto como desarrolladores o como usuarios de aplicaciones o programas, el factor tiempo es vital, es decir que no podemos darnos el lujo de esperar que un programa o aplicación se quede bloqueada o “congelada” (ya que sólo pueden ejecutar una acción a la vez) a la espera del resultado deseado, esto es bastante desagradable e incómodo. Para solucionar o evitar lo mencionado anteriormente, Java soporta la programación multihilos, es decir muchos hilos, tenemos que saber que un hilo es la unidad más pequeña de procesamiento de una tarea, es como si fuese la “tajada” de un proceso, Java lo que hace es una sincronización de los hilos, para que trabajen en paralelo, por ejemplo, un hilo se puede encargar de los eventos de la interfaz gráfica del usuario (interacción táctil con el usuario), mientras otro hilo puede mostrar una animación de fondo por pantalla, mientras otro realiza cálculos de la operación deseada.

2.3.1 FUNCIONAMIENTO DE LA MÁQUINA VIRTUAL JAVA

Nuestro procesador del ordenador sólo entiende de 0’s y 1’s, lo que se conoce como lenguaje máquina, y esto es muy complicado por no decir imposible de aprender para el ser humano, es por eso que existen los lenguajes de programación y junto con ellos los llamados compiladores, que juntos son dos herramientas muy potentes que nos facilitan las tareas de programación, pero java fue más allá, añadió un paso intermedio, añadió los bytecodes (el cual es generado por el compilador java), solo entendibles por la máquina virtual java, y luego estos podían ser ejecutados en la máquina virtual java, cabe resaltar que los programas java no se ejecutan en nuestro sistema operativo, sino en máquina virtual de java, la cual simula un sistema operativo virtual.

La máquina virtual java, es un programa nativo de la plataforma java, esto quiere decir que es un programa específicamente ejecutable en java, esto quiere decir que dicha máquina virtual no funciona en ninguna plataforma que no sea java,

Las máquinas virtuales java al igual que los microprocesadores reales y sistemas operativos actuales contienen información técnica detallada muy extensa que abarcaría para muchas páginas su estudio, sin embargo todo eso se puede simplificar diciendo que la MVJ contiene zonas de memoria y registros internos, cabe recordar que un registro es una memoria de muy poca capacidad pero con mucha velocidad de procesamiento, que nos permite almacenar y acceder datos que son usados frecuentemente, como un ayudante o auxiliar de la memoria principal.

En este subapartado no entraré en detalle sobre los registros de la máquina virtual java (MVJ), ya que son los típicos registros de cualquier prototipo de microprocesador, por el contrario sí que lo haré en las zonas de memoria de las que posee y gestiona la máquina virtual java ya que desde mi opinión personal es algo muy suyo de java, aunque también es cierto que su desconocimiento no impedirá que nosotros programemos en java, solo nos va a servir para obtener un conocimiento global de la gestión de memoria de la MVJ.

La máquina virtual java va a disponer de tres zonas de memoria: la zona de datos, heap y stock.

ZONA DE MEMORIA DE DATOS: Esta zona de memoria es la que no cambia en todo el proceso de ejecución del programa java en cuestión, también es llamada memoria inmutable, es en esta zona donde se almacenan las instrucciones de programa, las clases con sus metidos que hayamos creado y las constantes. Este es espacio es muy justo, es decir que es imposible que se reserve memoria más que lo necesario, ya que la cantidad exacta de memoria se conoce en tiempo de compilación.

ZONA DE MEMORIA STACK: En esta zona de memoria se guardan las referencias a objetos, es decir en los que usamos la palabra **new,** en este caso lo que se guarda no es un objeto sino una dirección de memoria, o dicho de una forma poco fina , es un puntero que apunta a un objeto, la expresión anterior es quizás inadecuada, pero creo que es la mejor forma de explicar, en java no se opera aritméticamente con los punteros como en C++, ya que como muy bien se sabe, la teoría nos dice que java no tiene punteros (en lugar de ello se usa el (término referencia). En esta zona de memoria también se guardan las variables de tipo primitivo.

ZONA DE MEMORIA HEAP: Esta memoria es dinámica, en esta se almacena los objetos creados.

Algo que suele ser común en java es el hecho de confundir lo que es un objeto y una referencia a objeto, el primero es lo que ya he explicado (un objeto es el ejemplar de una clase) el cual está en la memoria HEAP y una referencia a un objeto es una dirección de memoria, o lo que es lo mismo un número hexadecimal y se guarda en el STACK, el cual es una memoria pequeña y de rápido acceso.

Unas observaciones importantes entre las memorias HEAP Y SATCK, es que la HEAP (también conocida como montículo), es una memoria dinámica esto quiere decir por ejemplo irá creciendo conforme se vayan creando, además esta memoria es única, no hay replicas ni nada por el estilo y a diferencia de lo que se cree también es capaz de almacenar referencias a objetos (punteros de memoria).

Por el contrario la memoria STACK, es estática y no se modifica durante el desarrollo del programa, es una zona de memoria que se asigna a cada hilo (con lo cual hay varias zonas de memoria STACKS) y como mencioné anteriormente en ella se almacenan las variables locales y las referencias a objetos.

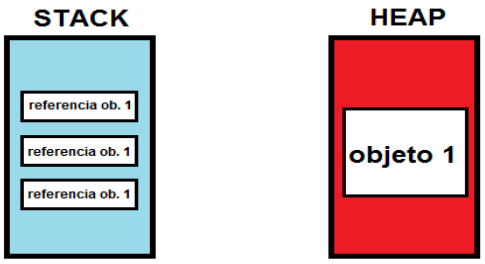


Figura 2-4 Zonas de memoria de la máquina virtual JAVA.

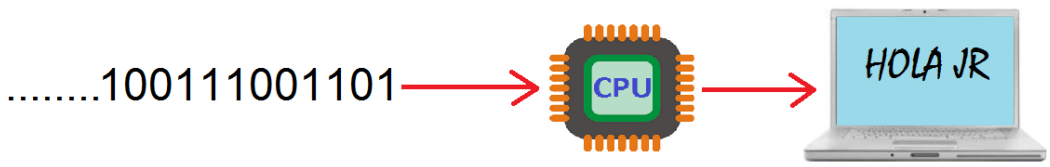


Figura 2-5 Comunicación con la cpu por medios dígitos binarios.

2.3.2 COMPILACIÓN Y EJECUCIÓN DE UN PROGRAMA EN JAVA

Nosotros vamos a escribir un código (escrito en lenguaje Java), y lo vamos a guardar en un archivo con una extensión **.java**, luego éste archivo se va a compilar (usando el compilador de Java) y lo que hace el compilador de java es transformar ese código que hemos escrito, en un formato binario llamado **bytecode,** que es como una especie de lenguaje máquina (lenguaje que la máquina virtual de java pueda entender) resultando un archivo con una extensión **.class** y es precisamente ese archivo el que vamos a poder ejecutar en cualquier sistema operativo que tenga una máquina virtual Java (la máquina virtual de Java es la que se encarga de traducir y ejecutar esos bytecodes teniendo como resultado un programa en sí). A continuación muestro un esquema de todo lo explicado anteriormente.

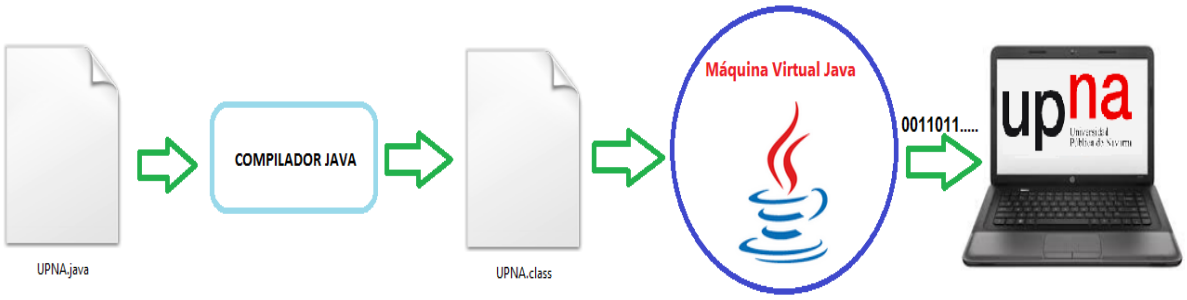


Figura 2-6 Esquema de compilación y ejecución de un programa en Java.

2.3.3 HAZLO UNA VEZ Y EJECÚTALO EN TODAS PARTES

Sólo necesitamos hacer el código y compilarlo una vez, es decir producimos el archivo .class, y ése archivo se va a poder ejecutar en todas las computadoras y ambientes que queramos, siempre y cuando cuenten con una máquina virtual Java, en el esquema inferior se puede apreciar el típico programa de todo lenguaje de programación que imprime por pantalla la archiconocida frase **"Hola Mundo"**, el cual lo compilamos, dando lugar a los bytecodes, con una extensión **.class**, los cuales pueden ejecutar en cualquier ordenador, con cualquier sistema operativo que tenga instalado la máquina virtual Java (MVJ).

En resumen lo que está compilado lo podemos ejecutar en todo ordenador que tenga la MVJ.

Figura 2-7 Logo de Java.

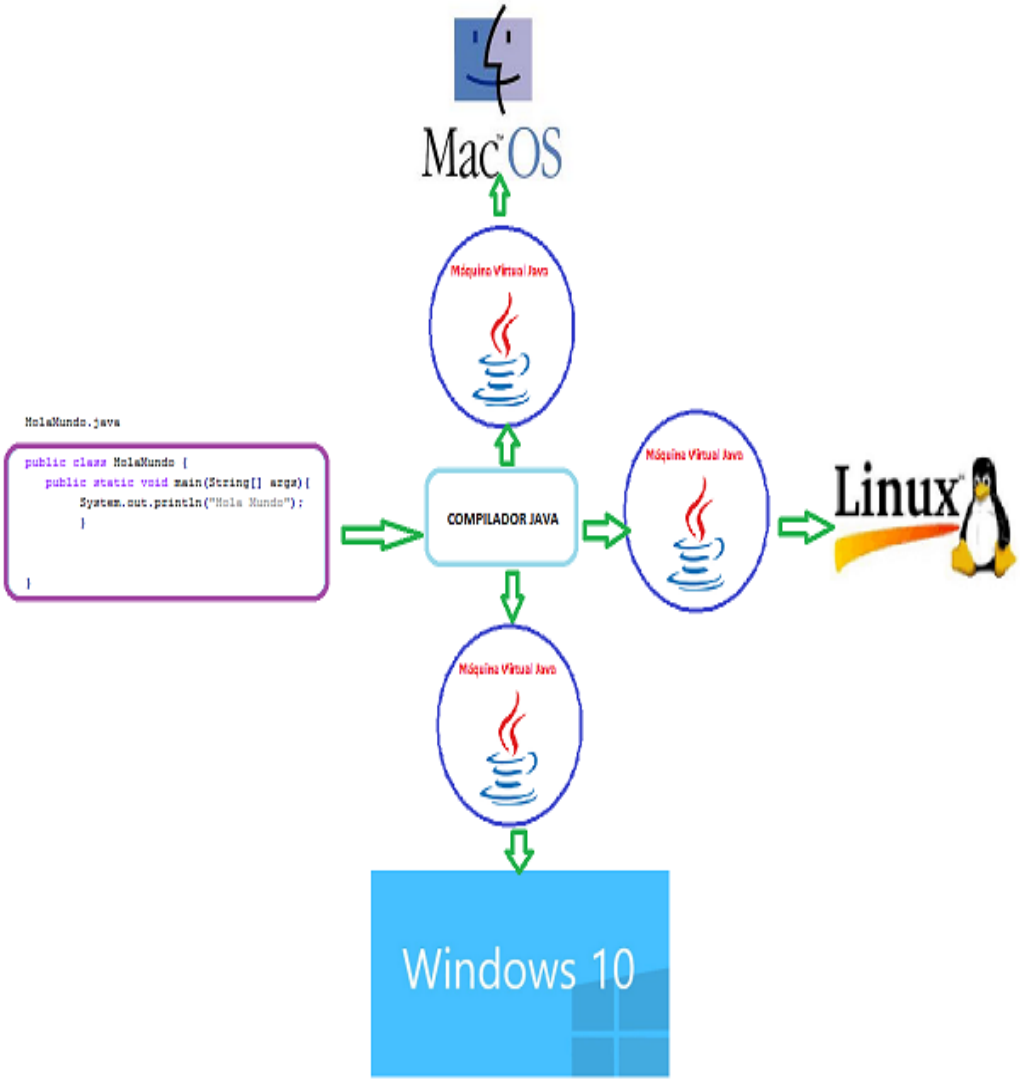


Figura 2-8 Esquema de compilación y ejecución de un programa Java en distintos sistemas operativos.

CAPÍTULO 3. ADQUISICIÓN DE DATOS DE TEMPERATURA

Código en Anexo 2.

Para la medición de los valores de temperatura en el interior de la cámara climática se ha utilizado el sensor digital SHT15 de la compañía Sensirion, realmente es un doble sensor, ya que nos permite medir temperatura y humedad relativa, dicho sensor se ha programado en el software Arduino, utilizando el lenguaje de programación del mismo (Arduino).

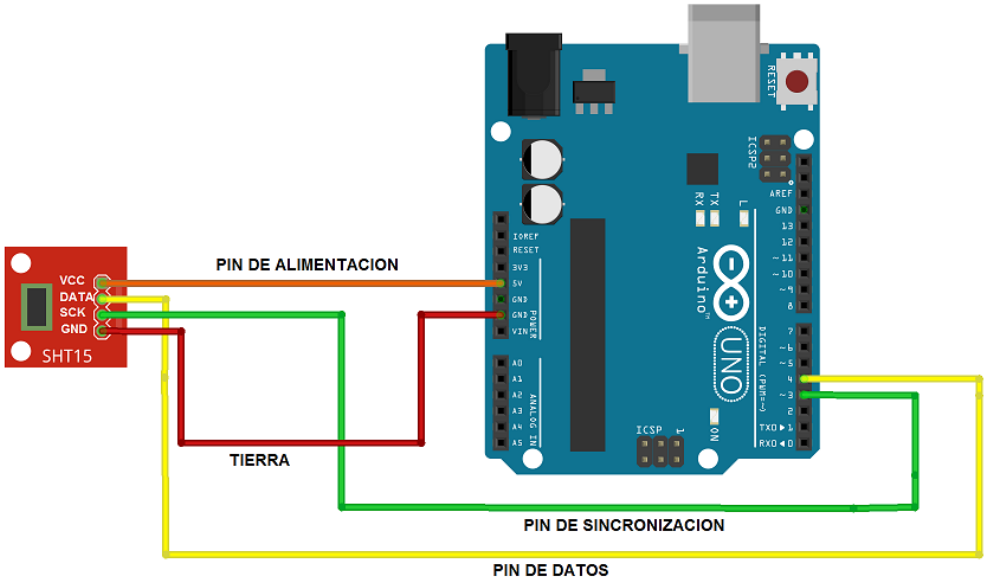


Figura 3-1 Esquema de conexión Arduino-Sensor SHT15.

La programación de este sensor fue muy laboriosa, ya que en un primer momento se pensó en utilizar la librería "Wire" de Arduino, dando por hecho que el microcontrolador (Arduino) y el sensor se comunicarían a través del protocolo I2C (también conocido con el nombre de TWI – de "**TW**o-w**I**re", es decir "dos cables") pero desafortunadamente, no fue así, ya que en el datasheet del sensor se especifica bien claro que éste no hace uso del protocolo I2C, por lo que dicho sensor se tuvo que programar estudiando toda la información de su datasheet, sus propio protocolo de comunicación y sus propias fórmulas a la hora de calcular la temperatura y humedad. Podemos acceder al datasheet del sensor en:

https://www.sparkfun.com/datasheets/Sensors/SHT1x_datasheet.pdf.

Como se puede apreciar en el esquemático de la Figura 3-1, el sensor SHT15 se comunica con el Arduino, mediante una línea de datos y otra de reloj, cabe mencionar que dicho sensor tiene cuatro pines que detallaré en breve.

3.1 ESPECIFICACIONES DE LA INTERFAZ

A continuación se muestra una tabla con la configuración de los pines del SHT15:

Pin	Name	Comment
1	GND	Ground
2	DATA	Serial Data, bidirectional
3	SCK	Serial Clock, input only
4	VDD	Source Voltage
NC	NC	Must be left unconnected

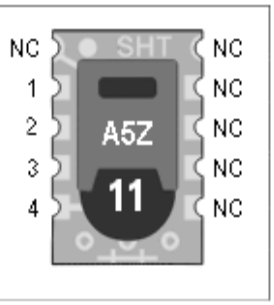


Figura 3-2 Pines del sensor SHT15.

Fuente: https://www.sparkfun.com/datasheets/Sensors/SHT1x_datasheet.pdf

3.1.1 PINES DE ALIMENTACIÓN (VDD y GND)

El sensor SHT15 requiere una tensión de alimentación (VDD) entre 2.4 y 5.5 V, después de alimentarlo éste necesita un tiempo de 11ms para estabilizarse, es decir no se debe enviar ninguna orden antes de este tiempo. También se necesita de un condensador de desacoplo de 100 nF entre la alimentación y tierra.

A continuación se muestra un circuito típico de aplicación (microcontrolador y sensor), incluye una resistencia pull-up, Rp, y el desacoplamiento entre VDD y GND por un condensador (mencionado anteriormente):

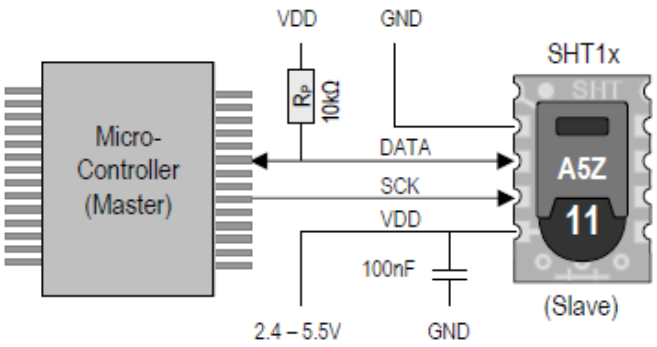


Figura 3-2 Circuito típico microcontrolador-sensor SHT15.

Fuente: https://www.sparkfun.com/datasheets/Sensors/SHT1x_datasheet.pdf

3.1.2 PIN DE ENTRADA DE RELOJ (SCK)

Este pin, es un pin de entrada, por este pin, el sensor va a recibir la señal de reloj que envía el microcontrolador del Arduino, esta señal de reloj sirve para sincronizar la transmisión entre el microcontrolador y el sensor.

3.1.3 PIN DE DATOS (DATA)

Este pin es bidireccional, es decir que por este pin el sensor va a recibir las instrucciones que le envía el microcontrolador del Arduino, y por éste mismo pin, el sensor envía datos al Arduino.

El pin DATA del sensor es un pin triestado, es decir tienes tres estados lógicos: 0, 1, y Z (alta impedancia, siendo traducido éste último estado, como una desconexión eléctrica del pin data), es por eso que se necesita de una resistencia de polarización a VDD (resistencia de pull- up), ya que sin esta resistencia la medida del sensor sería errónea.

3.2 COMUNICACIÓN CON EL SENSOR SHT15

Una vez descrito los pines del SHT15 y por ende las líneas de transmisión y sincronización presentes en la comunicación entre el micro y el sensor, ahora se va proceder a describir como es la “manera “en la que el microcontrolador se comunica con el sensor.

3.2.1 SECUENCIA “TRANSMISSION START”

Lo primero que debemos hacer, es enviar desde el Arduino al sensor una secuencia conocida como “Transmission Start” o “Inicio de Transmisión”, que no es más que una secuencia protocolaria, es decir una secuencia que tiene que ser enviada sí o sí, para que podamos dar inicio a la comunicación con el sensor.

Esta secuencia consiste en poner a nivel bajo la línea de datos mientras la línea de reloj permanece en alto seguida de un pulso a nivel bajo en la línea de reloj y luego un nuevo flanco ascendente en la línea de reloj y una subida en la de datos mientras la de reloj permanece en estado alto. Esto se puede ver mejor en la siguiente figura:

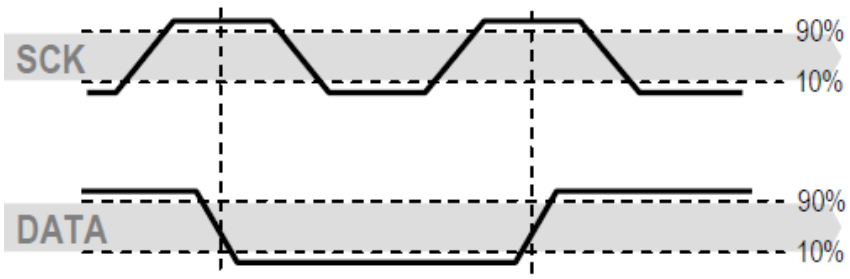


Figura 3-3 Secuencia de transmisión "Transmission Start"

Fuente: https://www.sparkfun.com/datasheets/Sensors/SHT1x_datasheet.pdf

3.2.2 COMANDOS

Después de realizar la secuencia de inicio ("Transmission Start"), se requiere de una secuencia de bits, a la que llamaremos "comando" que el microcontrolador debe enviar al sensor para obtener la medida de la temperatura y otras secuencias para distintas funciones

El protocolo de transmisión de los **comandos** que enviemos al sensor **están basados en un byte**, sabiendo que **los tres primeros bits de la izquierda serán siempre cero**, y **los cinco bits restantes son el comando en sí**.

Los distintos comandos disponibles se muestran en la siguiente tabla:

Command	Code
Reserved	0000x
Measure Temperature	**00011**
Measure Relative Humidity	**00101**
Read Status Register	00111
Write Status Register	00110
Reserved	0101x-1110x
Soft reset, resets the interface, clears the status register to default values. Wait minimum 11 ms before next command	**11110**

Figura 3-4 Tabla de los comandos del sensor SHT15.

Fuente: https://www.sparkfun.com/datasheets/Sensors/SHT1x_datasheet.pdf

A continuación muestro la misma tabla pero traducida:

Comando	**Código**
Reservado	0000x
Medida de temperatura	**00011**
Medida de la humedad relativa	**00101**
Leer Registro de estado	00111
Escribir registro de estado	00110
Reservado	0101x- 110x
Restablecimiento de software, restablece la interfaz, borra el registro de estado a sus valores predeterminados. Espere mínimos 11 ms antes del próximo mandato	**11110**

Tabla *Lista de los comandos del SHT15.*

El sensor SHT15 indicará que ha recibido correctamente el comando con un pulso de ACK (pulso de confirmación) en la línea de datos, que es bidireccional (poniendo o enviando un cero lógico en la línea de datos), pero lo hará después de recibir los ocho bits de comando y por supuesto después de recibir los ocho pulsos de reloj provenientes del Arduino, concretamente, el sensor enviará dicho ACK al Arduino, poniendo el pin de datos a nivel bajo (para ello el Arduino después que envía el comando al sensor, tiene que configurar su pin de datos, como entrada) tras el flanco descendente del octavo pulso de reloj.

NOTA IMPORTANTE: Por defecto la línea de datos se encuentra a nivel alto, y la línea de reloj o sincronización se encuentra a nivel bajo, y esto es así por la forma en que se encuentra polarizado el sensor.

6.2.3 SECUENCIA DE MEDIDA DE TEMPERATURA Y HUMEDAD RELATIVA

Dando por hecho que somos capaces de hacer que el Arduino mande un comando al sensor por ejemplo el “00000011” correspondiente a la medida de la temperatura (como se puede apreciar en la tabla de comandos), y una vez que el sensor envía el ACK al Arduino, éste debe esperar a que se complete la medida (es decir el tiempo de adquisición más el tiempo que tarda el sensor en entregar dicha medida al Arduino) entre 20 y 320ms, este tiempo no es exacto ya que depende de la resolución en bits de las medidas así como también de la alimentación y la velocidad del oscilador interno del

sensor. Para evitar esperas innecesarias por parte del Arduino, el sensor señaliza la conclusión de la medida, generando en la línea de datos un pulso bajo, de esta manera el Arduino sabe, que lo siguiente son datos válidos, para ello el Arduino lo que hace es comprobar, cada cierto tiempo, la línea de datos hasta encontrar un estado bajo en ella, si dicha línea de datos se encuentra en estado alto, quiere decir que aún no está lista la medida, y cuando reciba el Arduino un estado bajo en la línea de datos será sinónimo de que la adquisición de la medida ha sido completada y se pasará a leer dicha medida del sensor. Para lo cual ahora el Arduino genera ocho pulsos de reloj y guardará cada bit de información que envía el sensor, ahora bien, la trama que envía el sensor se compone de tres bytes, el primero de ellos correspondiente a los más significativos o MSB, el segundo a los menos significativos o LSB y el tercer byte corresponde a CRC (comprobación de redundancia cíclica), los datos al Arduino le llegan de izquierda a derecha, es decir primero le llega el byte MSB, después el byte LSB y por último el byte CRC (aunque éste byte no lo va a recibir el Arduino ya que como nos lo dice el datasheet este byte CRC es opcional), entonces el Arduino va a recibir dos bytes, y por cada byte que reciba tiene que enviar una confirmación o ACK (poniendo la línea de datos en estado bajo), pero como el Arduino va a ignorar el último byte correspondiente al CRC, lo que hace es enviar una anti confirmación o NACK (estado alto en la línea de datos) después de recibir el segundo byte, correspondiente al LSB. Luego de esto se da concluida la comunicación.

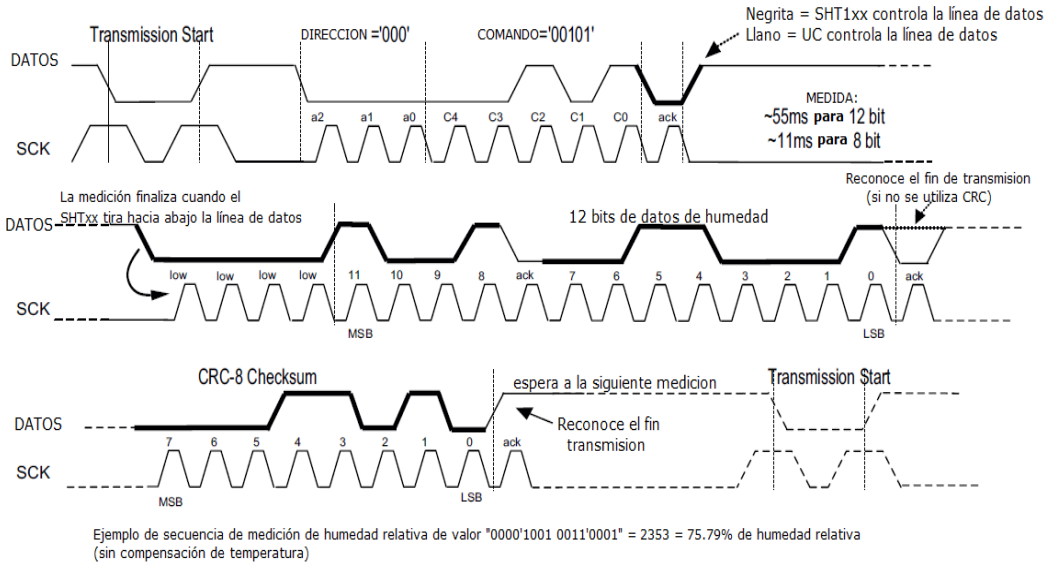


Figura 3-5 Secuencias de comunicación entre Arduino y el sensor SHT15.

NOTA IMPORTANTE: El sensor devuelve un valor que representa una humedad o temperatura, pero debemos de saber que no devuelve directamente la humedad relativa en porcentaje o la temperatura en grados centígrados. Los valores reales de temperatura y humedad han de ser calculados a partir de las fórmulas que se indican en el datasheet.

- $RH_{linear} = c_1 + c_2 \cdot SO_{RH} + c_3 \cdot (SO_{RH})^2 \quad (\%RH)$

SO_{RH}	C1	C2	C3
12 bit	-4	0.0405	$-2.8*10^{-6}$

- $T = d_1 + d_2 \cdot SO_T$

VDD	$d_1(^0C)$	$d_1(^0f)$
5V	-40.00	-40.00

SO_T	$d_2(^0C)$	$d_2(^0f)$
14bit	0.01	0.018

Para este proyecto final de carrera, sólo va a importar la medida de la temperatura:

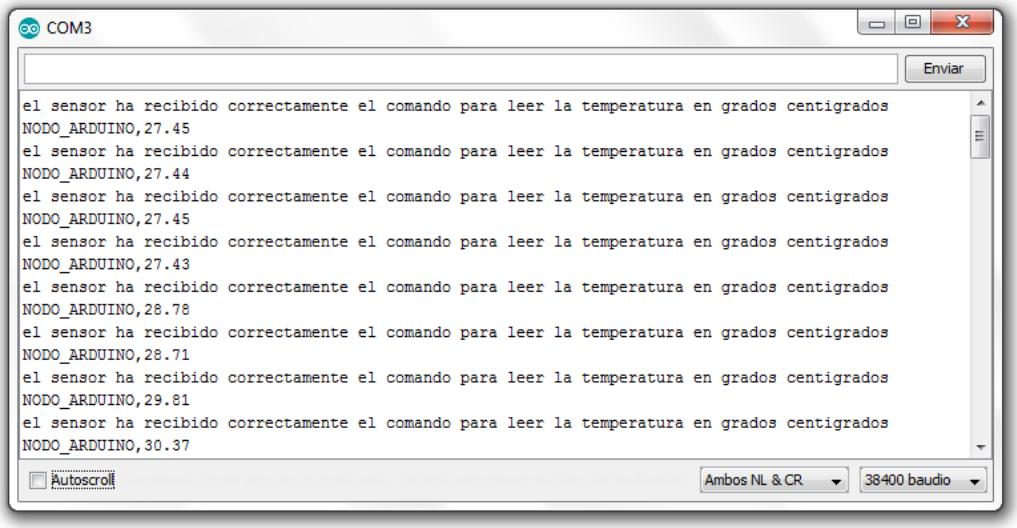


Figura 3-6 Monitor Serie, mostrando los valores de temperatura en tiempo real.

A continuación se muestra unas imágenes en las que se aprecia cómo están conectados el sensor y el Arduino UNO:

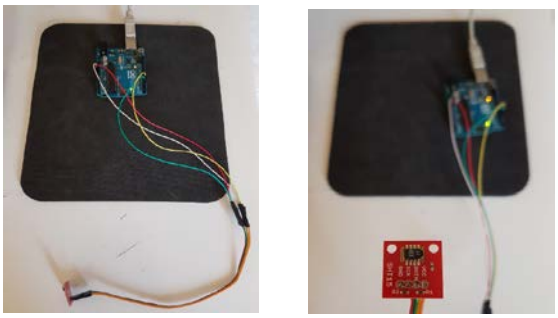


Figura 3-7 Conexión entre Arduino y el sensor digital SHT15 de Sensirion.

Y para terminar, se muestra un diagrama de flujo, que resume todo lo explicado anteriormente:

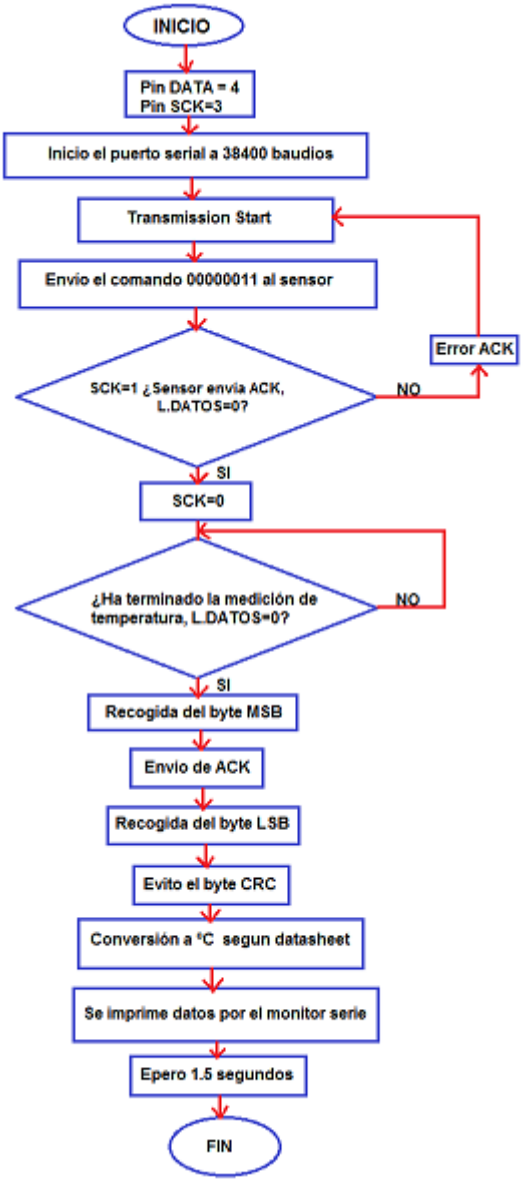


Figura 3-8 Diagrama de flujo, en el que se muestra todo el proceso de comunicación entre el microcontrolador del Arduino y el sensor digital SHT15.

CAPÍTULO 4. MÓDULO BLUETOOTH HC-05

Cabe mencionar que la placa Arduino (se dispone para este proyecto, del Arduino UNO y el Arduino MEGA, pudiendo elegir cualquiera de ellos) por sí sola no dispone de conectividad Bluetooth, pero si lo que queremos es dotar a dicha placa de la capacidad de comunicarse vía Bluetooth, deberemos conectarle o acoplarle algún módulo receptor-transmisor Bluetooth.

La mayoría de estos módulos van a tener rol de esclavos, o mejor dicho van a funcionar como dispositivos esclavos, por lo que deberá ser el otro extremo o nodo de la comunicación (generalmente un ordenador o un teléfono móvil con capacidad Bluetooth) el que tenga el rol de maestro o que funcione como dispositivo maestro. En estos casos, siempre será el ordenador o el móvil quien inicie la conexión con la placa Arduino y no al revés. En las próximas líneas nos centraremos en el módulo HC-05.

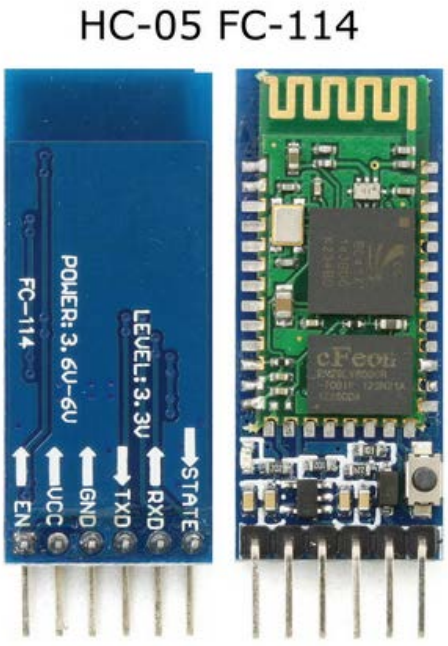


Figura 4-1 Módulo Bluetooth HC-05 versión FC-114.

Hay una gran variedad de módulos Bluetooth HC-05 en el mercado, en concreto el modelo que utilizo es el HC-05 FC-114, pero en términos generales cualquier modelo del HC-05 nos valdría.

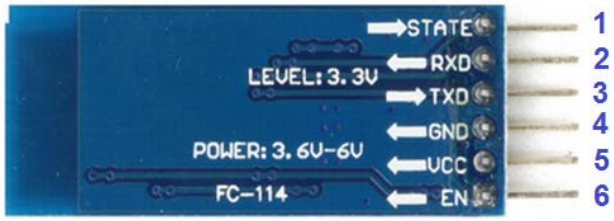


Figura 4-2 Pines del módulo bluetooth HC-05 FC-114.

Este módulo HC-05, dispone de seis pines, los cuales se detallará a continuación:

- **PIN STATE:** A decir verdad, existe muy poca información sobre este pin de estado, por lo tanto dicho pin no se utiliza, se supone que dicho pin, nos va a permitir obtener un estado del funcionamiento interno del módulo, pero en la práctica no se utiliza.
- **PIN DE RECEPCIÓN:** Por este pin, recibe datos del Arduino.
- **PIN DE TRANSMISIÓN:** Por este pin, envía datos al Arduino.
- **PIN GND:** Este el pin de tierra, va conectado a 0 voltios.
- **PIN VCC:** Es el pin de alimentación, debemos tener cuidado con el rango de voltajes que admite el módulo, en mi caso se alimenta el módulo con 5 voltios.
- **PIN EN O KEY:** Alimentando este pin con 5 voltios, vamos a poder configurar nuestro módulo bluetooth, es decir que el módulo entra en el modo de configuración, en el cual se van a poder configurar y visualizar, la velocidad de transmisión entre el arduino y el modulo, el nombre, la contraseña, el tipo de rol (maestro o esclavo), la dirección Mac y demás atributos propios de nuestro módulo Bluetooth a través de unos comandos llamados COMANDOS AT. En el caso de que este pin no reciba ningún voltaje de alimentación, el módulo bluetooth entra en modo usuario, es decir que está listo para enviar y recibir información del exterior a través de Bluetooth.

NOTA IMPORTANTE: Los módulos bluetooth HC-01, HC-03 y HC-05 (números impares), tienen dos roles es decir que pueden funcionar como maestros y como esclavos (lo cual es muy interesante). Por otro lado, los módulos HC-02, HC-04 y HC-06 (números pares) sólo tienen el rol de esclavos.

4.1 CONFIGURACIÓN DEL MÓDULO HC-05

Código en Anexo 2.

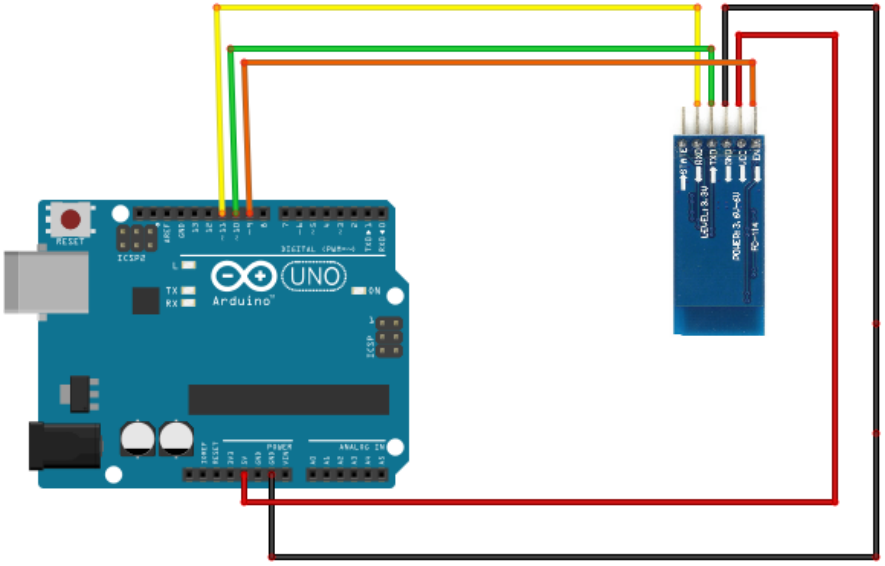


Figura 4-3 Esquemático del módulo bluetooth HC-05 y Arduino UNO.

En éste esquema se aprecian las conexiones que se han llevado a cabo, se pueden ver: el cable rojo de alimentación (5 V), el cable negro de tierra (0 V), y en la parte superior del esquemático vamos a hacer uso de los pines digitales del Arduino (10, 9 y 11), se puede apreciar como que el cable amarillo conecta el pin 11 del Arduino con el pin RXD (recepción) del módulo bluetooth, es a través de este cable que el arduino envía datos a dicho módulo, tenemos también un cable verde que conecta el pin 10 del Arduino con el pin TXD (transmisión) del módulo bluetooth, es a través de este cable que el módulo HC-05 envía datos al Arduino y por ultimo tenemos el cable de color anaranjado, el cual nos va a servir para comunicar el pin 9 del Arduino con el pin Key del módulo Bluetooth, es por medio de éste cable que alimentamos al pin KEY, permitiéndonos así entrar en modo configuración del HC-05. Una vez hecha las conexiones es necesario cargar el sketch de configuración (programa Arduino) en nuestra placa Arduino. Dicho sketch es muy necesario ya que tenemos que asignar ciertas funciones a los pines utilizados del Arduino, (con el cableado no basta, es necesario introducir lógica a los pines), además este sketch nos va a servir para hacer uso de los tan conocidos COMANDOS AT, y poder así configurar nuestro módulo bluetooth HC-05.

Algo muy importante a tener en cuenta es que el Arduino para poder comunicarse con otros dispositivos como el ordenador o el módulo bluetooth necesita de puertos series, los cuales sirven para establecer una comunicación serial, es decir bit a bit, por desgracia el Arduino UNO solo posee un puerto serie, que es el que se utiliza para comunicarse con el ordenador, de hecho es por este puerto serie que el ordenador alimenta al Arduino con lo que no se dispone de otro puerto serie para comunicarse con el modulo bluetooth, una opción sería elegir otra placa Arduino que disponga de varios puertos seriales (como la otra placa, el Arduino Mega), pero hay una solución incluso más sencilla, la cual es hacer uso de una librería llamada SoftwareSerial, cuya función es la de simular o mejor dicho crear puertos series virtuales, en nuestro caso dicho puerto serie virtual se ve reflejado en los pines digitales 10 y 11 del Arduino Uno, concretamente el pin 10 es el pin de recepción y el 11 es el pin de transmisión del Arduino, con lo que ahora ya se dispone de otro puerto serie, exclusivo para la comunicación entre el Arduino Uno y el módulo Bluetooth.

A continuación se muestra una captura del programa que es necesario cargar en Arduino para dar paso a la configuración del módulo Bluetooth:

Figura 4-4 Sketch Arduino de configuración del módulo Bluetooth.

Una vez de haber cargado éste programa en Arduino, procederemos a abrir el monitor serie del arduino:

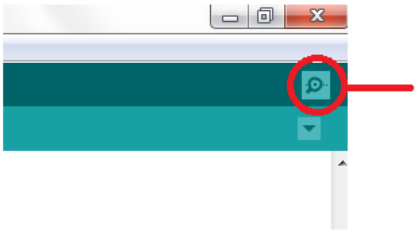


Figura 4-5 Símbolo del Monitor Serie del Arduino.

Y se nos abrirá la siguiente ventana:

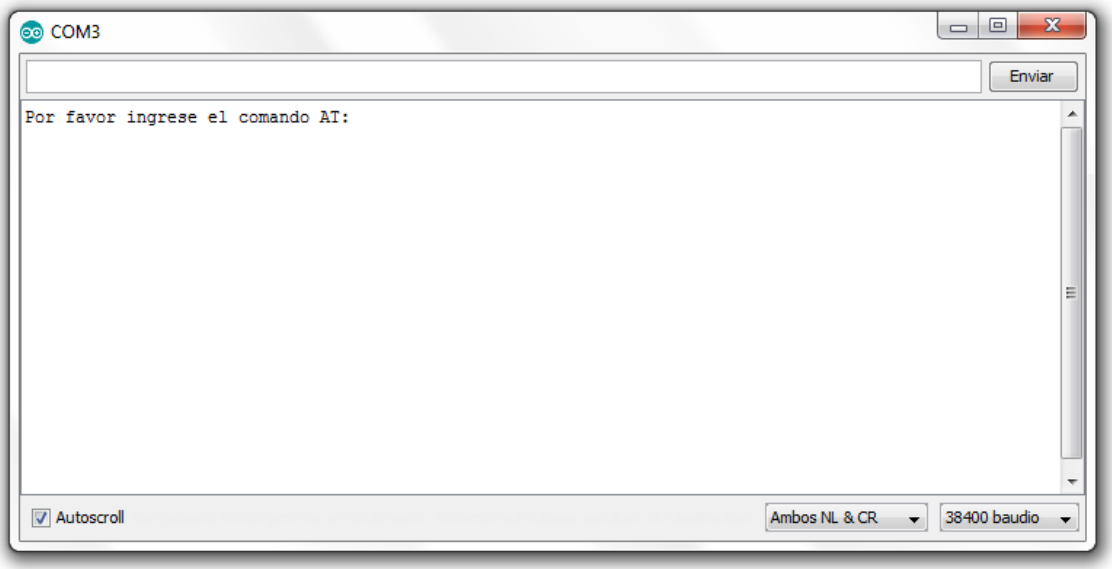


Figura 4-6 Comienzo de comandos AT en el Monitor Serie del Arduino.

A partir de aquí podemos ingresar todos los comandos AT que queramos, siendo los más conocidos:

```
AT      ----------->  RETORNA LA RESPUESTA OK
AT+NAME? ----->  MUESTRA EL NOMBRE ACTUAL
AT+NAME=MODULO BLUETOOTH JUNIOR ------------> CONFIGURO EL NOMBRE
AT+PSWD? ------>  MUESTRA LA CONTRASEÑA
AT+PSWD=0701
AT+ROLE?-------> MUESTRA EL ROL DEL MODULO (MAESTRO/ESCLAVO)
0=ESCLAVO
1=MAESTRO
AT+ADDR?------> MUESTRA LA DIRECCION MAC DEL MODULO
```

Hay muchos más comandos, pero con los citados es más que suficiente para configurar el módulo bluetooth HC-05:

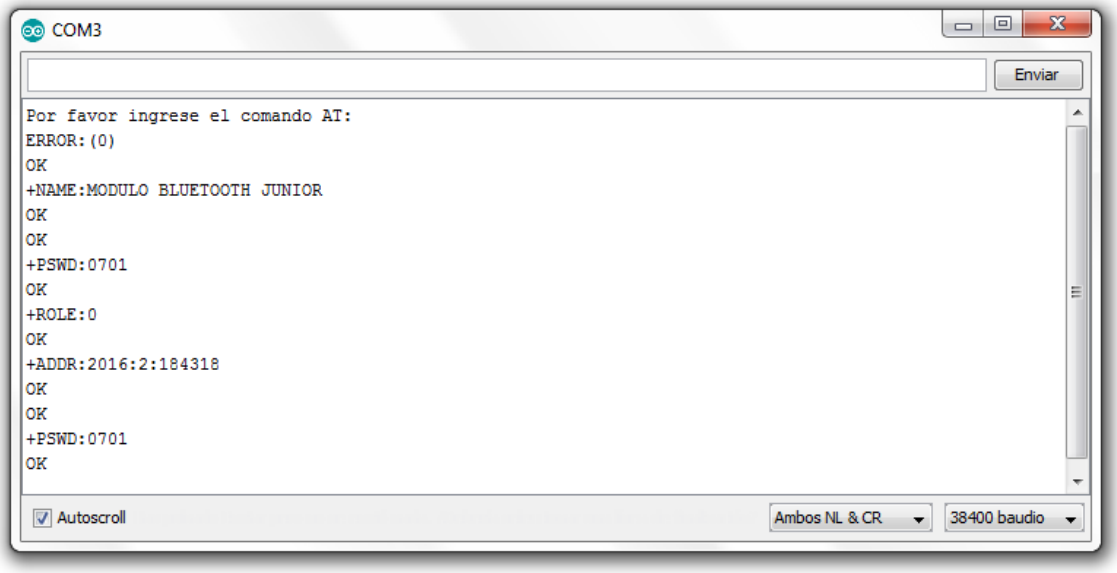
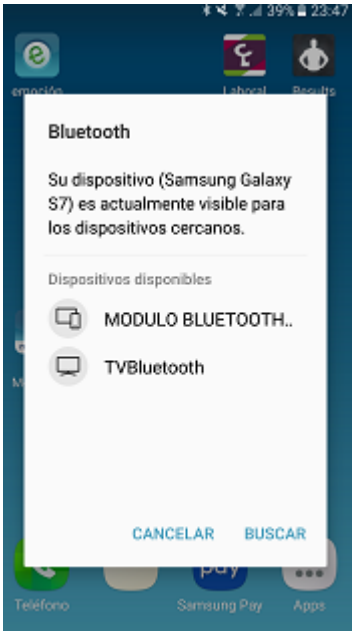


Figura 4-7 Configuración del módulo Bluetooth HC-05, haciendo uso de los comandos AT.

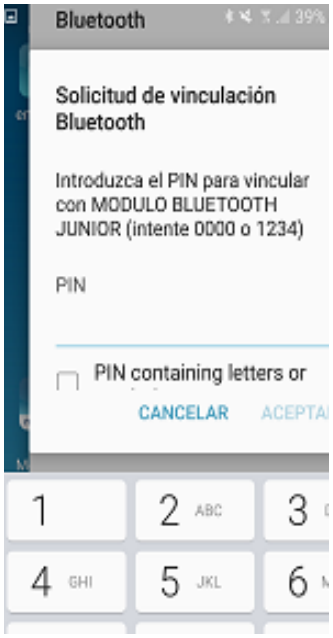
4.2 EMPAREJAMIENTO DEL MODULO HC-05 CON ANDROID

Habiendo configurado el HC-05 de una forma personal, ahora se procederá a mostrar unas imágenes en las cuales el dispositivo Android se vincula o empareja con el módulo HC-05:



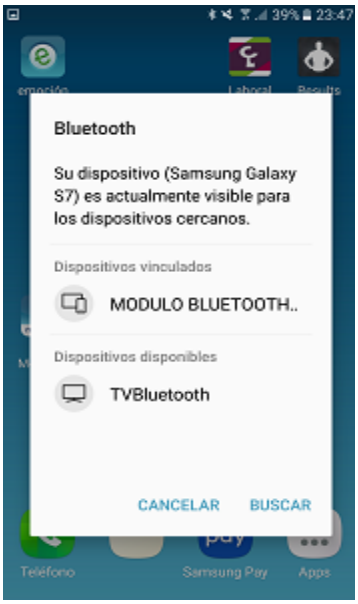
El dispositivo Android, activa bluetooth y detecta un dispositivo con el nombre MODULO BLUETOOTH JUNIOR

Figura 4-8 Primer paso de emparejamiento entre el dispositivo Android y el módulo HC-05.



Cuando se configuró el módulo Bluetooth, se le puse una contraseña, la cual era 0701. Entonces para que el móvil Android se vincule con el HC-05, se tiene que escribir en éste, esa misma contraseña.

Figura 4-9 Segundo paso de emparejamiento entre el dispositivo Android y el módulo HC-05.



Finalmente ambos dispositivos quedan vinculados.

Figura 4-8 Tercer y último paso de emparejamiento entre el dispositivo Android y el módulo HC-05.

CAPÍTULO 5. PROGRAMACION ORIENTADA A OBJETOS (POO)

Hasta ahora se ha tratado todo lo relacionado al sensor digital SHT15, placa Arduino y módulo Bluetooth HC-05. Puesto que toda aplicación Android (uno de los objetivos de este proyecto, es desarrollar una aplicación Android, que reciba los datos que son enviados por el módulo bluetooth HC-05, pero de esto, se hablará en los últimos capítulos) está escrita en Java, y ésta a su vez es un lenguaje de programación orinetada a objetos, y tal como se mencionó en el capítulo 2, la programación orientada a objetos (POO), es una forma distinta y especial de ver la programación, acercándonos coloquialmente a la vida real mediante conceptos o términos que son muy fáciles de comprender (dichos conceptos se detallarán en breve).

Con la POO, tenemos que escribir nuestros programas pensando en términos de objetos, clases, métodos y demás conceptos que se explicará continuación.

5.1 ¿CÓMO SURGE LA POO?

Durante mucho tiempo los programadores han seguido una forma de programar denominada **programación estructurada**, la cual surgió a finales de 1970, cuya característica principal está en la secuencia de las instrucciones.

La programación estructurada contiene únicamente subrutinas y estructuras de control, cabe recordar que una **subrutina** es una porción de código que forma parte de un programa, y que tendrá por **"misión**" realizar una tarea específica, independientemente del resto de código del programa que se esté realizando. Las **estructuras de control** que posee la programación estructurada son: secuencia (ejecución sucesiva de operaciones, según el orden en el que hayan sido escritas, dicho de otra manera; las instrucción u operación sigue en secuencia a otra), condicional o selección (se realiza una u otra acción dependiendo de una condición, mediante el uso de **if y switch**) e iteración (se repite una o más operaciones mientras se cumpla una condición, como ejemplo tenemos a los bucles **for y while**).

Éste modelo de programación es muy claro, todo está perfectamente ordenado y su secuencia es tal que los programas son fáciles de seguir o leer, la estructura de los programas es clara y además que el código del programa es reutilizable, pero el

inconveniente principal es que se obtiene un único bloque del programa, que suele ser muy grande con lo cual su manejo es medianamente problemático y visualmente tanto código en un solo bloque no es para nada agradable.

Además de lo mencionado anteriormente, hoy en día las aplicaciones informáticas son cada vez más sofisticadas (esto es obvio ya que no estamos en los orígenes de la programación estructurada de los 70, y las tecnologías evolucionan a una velocidad de vértigo) y lo son aún más en el ámbito de la telefonía móvil, donde cada vez nos encontramos a usuarios más exigentes, y sobre todo en el ámbito gráfico, donde la programación estructurada se queda muy limitada, y es el momento oportuno para dar paso a una nueva forma de programar llamada **programación orientada a objetos (POO).**

Con la programación orientada a objetos se adquiere una nueva **"filosofía"**, de cara a realizar nuestros programas, teniendo como pilar o plantilla de nuestro programa, algo que llamaremos clase y que junto a otros términos y definiciones que se explicará a continuación, serán todo lo necesario para entender la programación orientada a objetos, la cual desde un cierto punto de vista, no es difícil, lo que es difícil en sí, es "lanzarse" a programar en un lenguaje orientado a objetos sin siquiera entender bien los conceptos y términos esenciales (**"vitales"**) para entender la POO. Es por ello que el siguiente apartado estará destinado a detallar dichos conceptos y términos.

5.2 ACLARANDO TERMINOS Y DEMAS CONCEPTOS DE LA POO

Sin lugar a dudas el primer término que se tiene que entender es el de **clase,** y como ya se mencionó en más de una vez, es el **pilar de toda programación orientada a objetos (POO),** dicho de una manera simple se dirá que es la idea generalizada de algo, a partir de la cual se desarrollan otras ideas a la que llamaremos **objetos**, y a su vez cada objeto tendrá unas propiedades que llamaremos **atributos** y unos comportamientos a los cuales llamaremos **métodos**, que no serán más que unas operaciones (o acciones) disponibles específicas del objeto.

Al hablar sobre **programación orientada a objetos (POO)** es más que evidente que se tiene que hacer hincapié en el término **objeto**, el cual es un **"ejemplar"** del conjunto de objetos que comparten características similares definidas en la **clase**. A modo de

ejemplo podríamos definir una **clase** llamada **Universidades de España,** y un **objeto** de dicha clase podría ser **UPNA,** otros objetos serian: Universidad Complutense de Madrid, Universidad de La Rioja y Universidad de Mondragón, como se puede ver hay cuatro objetos (ejemplares), de la clase **Universidades de España,** con éste ejemplo podemos ver la simplicidad de los términos **clase y objeto,** pero esto no es todo, como se dijo anteriormente cada objeto va a tener unos **atributos y métodos**, los cuales tienen que estar definidos en la clase del objeto que pertenecerá a dicha clase, para aclarar dichos términos se pondrá un ejemplo, que será tan fácil de entender como el anterior, ahora pensemos en una **clase** como puede ser **vehículos de transporte**, y ahora se define una serie de **objetos** que pertenecen a dicha clase como puede ser **motocicleta, ciclomotor, coche, avión, bus y furgoneta.** En terminología de POO (programación orientada a objetos), diremos que **instanciar una clase** es lo mismo que **crear un objeto de ésa clase,** o lo que es lo mismo, **un objeto es una instancia de una clase**.

Cada uno de estos objetos de la clase vehículos de transporte tendrá una serie de **atributos** (propiedades) como puede **ser color, modelo, número de puertas, cilindrada, kilometraje, etc**. Como se acaba de ver, estos atributos son las características que posee cualquier vehículo de transporte, pero dichos vehículos también poseen una serie de **acciones** que llevadas a términos de **programación orientada a objetos (POO)** serán los denominados **métodos,** los cuales pueden ser: **arrancar, detenerse, girar a la izquierda, girar a la derecha, encender las luces y activar el aire acondicionado.**

Con estos ejemplos, ya se debe estar preparado para entender los términos básicos de la POO, como son: **clase, objeto, atributos y métodos,** no obstante nosotras las **personas,** también podemos ser modeladas en términos de la programación orientada a objetos, por poner un ejemplo, **Juniors Antonio Medina Landeón**, quien redacta éstas líneas, podría ser un **objeto** llamado **persona1,** que pertenecería a la **clase** llamada **personas,** y como miembro de ésta clase tendría unos **atributos** que serían: **edad, nombre, peso, altura y nacionalidad** (se podría poner otros atributos pero con los mencionados es más que suficiente) y también tendría unos **métodos,** que serían una serie de **comportamientos**, que tenemos en el día a día, o dicho de una mejor manera es una serie de **acciones que podemos realizar,** los cuales son: **hablar, dormir, reír,**

bailar, comer y conducir un coche, se podría añadir más métodos, pero con estos es más que suficiente.

Como podemos ver en este apartado de éste capítulo, términos tan complejos pueden volverse fácilmente entendibles si los modelamos con la realidad.

Se va a concluir este apartado con un resumen: la **clase** es el pilar de toda **programación orientada a objetos,** describe un conjunto de **objetos**, que serán **ejemplares** de dicha clase y tendrán propiedades (**atributos**) y comportamientos (**métodos**) similares.

5.3 OTROS TERMINOS Y DEMAS CONCEPTOS DE LA POO

Sin lugar a dudas los términos y conceptos más importantes fueron descritos en el apartado 5.2 en el cual nos adentramos en la **programación orientación a objetos (POO)**, por medio de ejemplos muy fáciles de entender, en éste apartado se explicará otros conceptos muy importantes como lo son: **herencia, encapsulamiento, polimorfismo, clases abstractas e interfaces**. Es conveniente advertir, que estos términos son más complejos que los vistos en el apartado anterior, es por ello que se abordará dichos conceptos mediante subapartados, intentando ser lo más claro posible.

5.3.1 HERENCIA

La **herencia** en programación orientada a objetos, es fácil de entenderla si se le asemeja con lo que se entiende por **herencia en la vida real**, la cual nos dice que **características o rasgos** vamos a obtener de nuestros **padres o abuelos,** nosotros como **hijos** queramos o no, nos parecemos a nuestros progenitores, y éstos a sus padres, es algo inevitable, y en la **programación orientada a objetos (POO)** es lo mismo, se puede crear clases que hereden de otras clases, es decir **clases hijas** que hereden de **clases padres,** en términos de la POO se dirá que una **clase** (hija) **extiende de otra clase** (padre), cuando ésta hereda atributos y métodos de la clase padre, no obstante además de los atributos y métodos heredados de la clase padre, las clases hijas tienen sus propios métodos y atributos (esto es obvio ya que en la vida real, nosotros los hijos no somos una copia exacta de nuestros padres). Cabe mencionar que la palabra clave **extends** es usada en la declaración de clases, para crear una clase hija de otra.

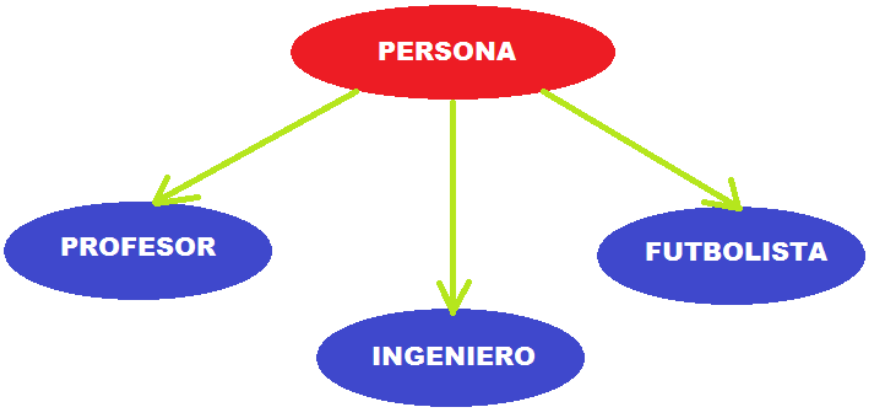


Figura 5-1 Esquema de una clase padre y sus clases hijas (azules)

En el esquema superior se puede apreciar la clase persona y sus tres clases hijas: profesor, ingeniero y futbolista, estas tres clases como tal no dejan de ser personas, es decir todas ellas tendrán atributos comunes como el nombre, el sexo y la edad. También compartirán métodos comunes como comer, dormir, caminar, etc. Y a su vez estas clases hijas tendrán atributos y métodos propios de ellas, es decir por ejemplo: los atributos y comportamientos de la clase futbolista serán muy diferentes de los de la clase ingeniero y profesor.

5.3.2 ENCAPSULAMIENTO

El **encapsulamiento** es uno de los conceptos más importantes en términos de la **seguridad del programa o aplicación** que estemos realizando, el propio **concepto encapsulamiento** es fácil de entenderlo por sí solo, cuando uno escucha o habla de **encapsular**, se nos viene a la mente la idea de la **medida en la que se quiere proteger algo**, y es justamente esa idea la que se plasma en el campo de la programación orientada a objetos, en la **POO medimos o clasificamos el rango de seguridad de métodos o atributos** de nuestros programas o aplicaciones mediante el uso de tres palabras reservadas: **public, protected y private.** A continuación se explicarán dichas palabras reservadas o claves y también el hecho de no poner ninguna de estas palabras (DEFAULT) a continuación:

PUBLIC: Mediante ésta palabra reservada, se accede a los atributos y métodos de una clase, desde otras clases y objetos (instancias) de éstas.

PROTECTED: Mediante ésta palabra reservada, se puede acceder a los atributos y métodos de una clase, desde las clases hijas de ésta.

PRIVATE: Mediante ésta palabra reservada, sólo se puede acceder a los atributos y métodos de una clase, desde la propia clase.

DEFAULT: En éste caso no hay ninguna palabra reservada, es decir ninguna de las tres estudiadas anteriormente, con esto lo que se consigue es lo siguiente: tenemos **una clase que pertenecerá a un paquete** (en breve se explicará que es un paquete), **dicha clase poseerá unos métodos y atributos**, y sólo **desde cualquier clase que pertenece a ese mismo paquete** se podrá acceder a dichos atributos y métodos.

Ahora bien, en java que es el lenguaje de programación orientada a objetos que se va a utilizar, el término paquete hace referencia al espacio de nombre, separado por puntos, en el cual se está organizado un conjunto de clases e interfaces relacionadas entre sí (toda la sintaxis JAVA de los apartados 5.2 y 5.3 serán explicados en el apartado 5.4). Éste concepto puede resultar difícil de asimilar, pero se puede asociar los paquetes a las diferentes carpetas que hay en nuestros ordenadores.

5.3.3 POLIMORFISMO

El polimorfismo es un concepto que va a estar muy ligado al de herencia, se debe recordar que cuando se habla de herencia, se hace referencia a la clase padre y clases hijas, y que éstas últimas heredan los atributos y métodos de su clase padre, hasta aquí todo bien, pero yendo un poco más allá, se podría definir el polimorfismo como una variable que puede tomar muchas formas (una variable puede almacenar diferentes tipos de objetos) y esto quiere decir que cuando uno crea un objeto (haciendo uso de la palabra reservada **new**) lo normal debería ser que ese objeto se almacene en una variable de la clase de dicho objeto, pero gracias al polimorfismo se puede alterar esta lógica, haciendo uso de la herencia, de la siguiente manera: creamos un objeto (perteneciente a una clase hija) pero ésta vez lo almacenamos en una variable de la clase padre.

En resumidas cuentas, una variable que es de una clase padre puede tomar la forma de cualquiera de sus clases hijas.

5.3.4 CLASES ABSTRACTAS

Hablar de clases abstractas puede parecer redundante ya que de por sí, una clase ya es abstracta, pero aún se podría volver más abstracta una clase, y esto se consigue haciendo que al menos uno de sus métodos no tengan desarrollo alguno, es decir que solo figure el nombre del método (sin llaves ni sentencias en el interior del método), en resumidas cuentas, al menos un método no está implementado, el cual será llamado método abstracto, y todas las subclases de la clase abstracta podrán implementar dicho método o métodos abstractos. Cabe resaltar que todos los métodos abstractos tienen que implementarse en las subclases, sería incorrecto implementar solamente los que resulten del interés de cada subclase.

5.3.5 INTERFACES

Una interfaz en java, es una clase que no se puede implementar, es simplemente una definición, es una abstracción de lo abstracto, y sirve para que una clase herede comportamientos (métodos) de diferentes interfaces (se tiene que recordar que en java no existe herencia múltiple), para que una clase herede los métodos de una interfaz es necesario que dicha clase implemente la interfaz en cuestión, mediante la palabra reservada **implements.**

En resumen una interfaz es una colección de variables y métodos no implementados, y toda clase que utilice o implemente una interfaz, está obligado a implementar cada uno de los métodos de la interfaz.

A menudo los programadores suelen confundir interfaz con clase abstracta, lo cual es entendible ya que ambas tienen métodos no implementados, pero se tiene que recordar que la principal diferencia entre ellas es que la interfaz no implementa ninguno de sus métodos, mientras que en una clase al menos uno de sus métodos no estará implementado.

5.4. EJEMPLIFICANDO TODO LO EXPLICADO EN LOS APARTADOS 3.2 Y 3.3

En este apartado por medio de ejemplos se va a intentar que todo lo explicado en los dos últimos apartados quede totalmente entendido, por medio de los siguientes ejemplos que tendrán la mayor claridad y sencillez posible.

5.4.1 CLASE

Una clase como se explicó, es una plantilla muy útil para la creación y definición de objetos, una clase va a tener un nombre, unas propiedades o atributos, y unos comportamientos o métodos.

```
   /* Se crea una clase que se llama ClasePrincipal */
public class ClasePrincipal {
    /* Se puede definir atributos que no son más que variables */
    public String atributo= "En la variable atributo escribo éste
texto, o cualquier otro";
    /* También se puede definir comportamientos que no son más que
métodos */
    public void metodo() {
    /* Aquí irán todas las instrucciones o sentencias, es decir la
implementación del método */
    }
}
```

5.4.2 OBJETO

Un objeto es producto de algo abstracto como lo es clase, pero a diferencia de ella, es más específico, aquí es donde los atributos y métodos definidos en la clase contenedora cobran sentido, cuando hablemos de objeto pensemos en una manzana o en un libro o cualquier cosa que podamos tocar, suena muy sencillo lo que se acaba de escribir, pero es que es la verdad. Una vez, el autor de estas líneas, leyó en un libro que las clases eran los moldes, y los objetos eran las galletas que se obtenían a partir de ellas. A continuación se verá en términos de java, como se declara y crea un objeto, así como la utilización de los métodos y atributos definidos en su clase.

```
   /** Vamos a ver todo lo explicado anteriormente por medio de
codificación java **/
   /* Se empieza declarando el objeto miObjeto de tipo Principal */
Principal miObjeto;
   /*Una vez definido, ahora toca crear el objeto, y esto se hace con
la sentencia new */
   miObjeto = new Principal();
   /* Ahora hacemos usos de los atributos y métodos de la clase */
miObjeto.atributo="Estoy modificando el nuevo valor del atributo";
miObjeto.metodo(); /* Hago una llamada al método metodo()  */
```

5.4.3 HERENCIA

Habiéndose explicado anteriormente lo que es la herencia, es decir lo relacionado a clase padre y clase hijas, y todo lo que ello conlleva, se debe recordar que en java solo podemos heredar de una sola clase padre y que se representa mediante la palabra reservada **extends.**

A continuación se muestra un ejemplo de herencia en codificación java:

```
    public class Animal {
    /** ATRIBUTOS **/
    public String tamaño; /* puede ser pequeño, mediano o grande */
    public int numero_patas;
       public String nombre;
       public String sexo;

    /** MÉTODO **/
    public void comer(){
        System.out.println("El animal esta comiendo");
    }
}
```

Se tiene una clase padre llamada Animal, que tendrá dos atributos y un método, los cuales se van a heredar a las clases hijas de Animal que veremos a continuación:

```
   public class Caballo extends Animal {
       /** ATRIBUTOS **/

    public String tamaño = "grande";
    public int numero_patas = 4;
       public int numero_dientes = 40; /* los caballos tienen 40
dientes */

       /** MÉTODOS **/

    public void comer(){
        System.out.println("El caballo esta comiendo");
    }
       public void correr (){
         System.out.println("El caballo es un animal muy veloz");
    }
}
```

La clase Caballo hereda de la clase Animal y esto se puede ver por el uso de la palabra **extends,** además hace uso de los atributos y método heredados, a su vez ésta clase hija tiene su atributo y método propios.

Se debe mencionar que aunque los métodos o atributos heredados no están definidos en la clase hija, se puede hacer uso de ellos siempre que queramos, ya que

aunque no esten definidos, sí estan heredados (fijense en los atributos **nombre** y **sexo** del objeto **cab**):

```
public class pruebaHerencia {
    public static void main(String[] args) {
       Caballo cab = new Caballo();
       cab.nombre = "Pegaso";
       cab.sexo = "macho";
       cab.comer();
      }
 }
```

Para terminar éste subapartado de herencia, se verá otra clase hija de Animal, que en este caso, aunque no se aprecie en la implementación de dicha clase los atributos y método heredados, siempre los va a poseer y hacer uso de ellos.

```
public class Ave extends Animal {
    /** ATRIBUTO **/
  public int numero_plumas;
    /** MÉTODO **/
     public void volar (){
       System.out.println("el ave esta volando");
    }
}
```

5.4.4 ENCAPSULAMIENTO

Como se explicó en el subapartado 5.3.2, el encapsulamiento abarca el tema de la seguridad de los datos de nuestro programa o aplicación que se esté desarrollando (cuando se habla de datos, se está refiriendo a variables y a métodos), para ello entran en juego los modificadores de acceso: public, protected y private que se van a encargar de establecer los permisos o niveles de visibilidad o acceso de nuestros datos.

```
public class Persona{

    /** ATRIBUTOS **/

  public String nombre;
  private String apellido1;
  protected String apellido2;
    public int edad;

    /** MÉTODOS PÚBLICOS**/

  public void hablar() {
      System.out.println("La persona ahora habla");
  }
  public void caminar() {
```

```
        System.out.println("La persona ahora camina");
    }
      public void comer() {
        System.out.println("La persona ahora come");
    }
      public void dormir() {
        System.out.println("La persona ahora duerme");
    }
      public void despertar() {
        System.out.println("La persona ahora despierta");
    }
      public void estudiar() {
        System.out.println("La persona ahora estudia");
    }
    public void trabajar() {
        System.out.println("La persona ahora trabaja");
    }

}
```

En esta clase se puede apreciar una variedad de permisos aplicados a los atributos de ésta, en cuanto a los métodos se los ha puesto públicos, aunque podrían ser de cualquier tipo sin ningún problema.

Sinceramente, el encapsulamiento conviene tratarlo más a fondo, ya que muchos piensan que éste tema, solo basta con lo explicado anteriormente, pero nada más lejos de la realidad, aún nos falta ver los métodos **sets y gets** que nos van a permitir manipular nuestros datos de forma segura (esto se explicará en este mismo subapartado pero más adelante).

Supongamos que se tiene una clase llamada Gato (con cualquier otro animal me valdría), la cual tiene unos atributos, todos de ellos públicos, con lo cual se podría acceder a ellos desde el exterior de su propia clase, concretamente desde una clase que he llamado clasePrincipal.

```
  public class Gato {

    /** ATRIBUTOS **/

    public String nombre;
    public double peso;
    public String color;

    /** MÉTODO**/

    public void imprimirPorConsola() {
        System.out.println();
        System.out.println("nombre: " + this.nombre);
```

```
        System.out.println("color: " + this.color);
        System.out.println("peso: " + this.peso);

    }
}
```

Si se observa bien, en las sentencias del método imprimirPorConsola aparece la palabra **this**, la cual se explicará al término de este subapartado, ahora se aprecia la clase clasePrincipal, la cual como su nombre indica es una clase principal, es decir es la clase más importante del proyecto Java que se esté realizando, ésta clase principal tendrá una función o método llamado **main** (la cual es obligatoria), no se mencionó antes, pero todo proyecto Java va a estar formado por varias clases, y más cuanto más complejo sea dicho proyecto.

```
public class clasePrincipal {

  public static void main(String[] args) {
      Gato gato1 = new Gato();
      gato1.nombre = "Turco";
      gato1.color = "amarillo";
      gato1.peso = 1.5; /* peso en kilogramos */
      gato1.imprimirPorConsola();

      Gato gato2 = new Gato();
      gato2.nombre = "Tom";
      gato2.color = "negro";
      gato2.peso = 2; /* peso en kilogramos */
      gato2.imprimirPorConsola();
  }

}
```

Como se puede ver desde la clase clasePrincipal se accede a los atributos de la clase Gato, así como se llama al método de la misma, sin ningún problema, lo cual es lógico ya que todos los modificadores de acceso de la clase Gato son public.

Ahora bien, si dentro de la clase Gato, se modifica el permiso de acceso de la variable nombre a private, ya no podríamos llamar a este atributo de forma externa como lo hicimos antes desde la clase clasePrincipal.

```
public class Gato {
    /** ATRIBUTOS **/

    private String nombre;
    public double peso;
    public String color;

    /** MÉTODO**/

    public void imprimirPorConsola() {
        System.out.println();
        System.out.println("nombre: " + this.nombre);
        System.out.println("color: " + this.color);
        System.out.println("peso: " + this.peso);

    }
}
```

Y ahora la clase clasePrincipal tal como está, ya no es de utilidad para poder acceder al atributo **nombre** de la clase **Gato**, ya que dicho atributo ahora es privado, y solo se puede acceder desde la propia clase Gato.

```
public class clasePrincipal {

    public static void main(String[] args) {
        Gato gato1 = new Gato();
        gato1.nombre = "Turco";
        gato1.color = "amarillo";
        gato1.peso = 1.5; /* peso en kilogramos */
        gato1.imprimirPorConsola();

        Gato gato2 = new Gato();
        gato2.nombre = "Tom";
        gato2.color = "negro";
        gato2.peso = 2; /* peso en kilogramos */
        gato2.imprimirPorConsola();
    }

}
```

Como se citó, ésta clase ya no nos sirve para acceder a un atributo privado, entonces es aquí donde los métodos SET y GET entran en acción, ya que gracias a ellos es posible, “burlar” la seguridad de atributos privados y acceder a ellos desde una clase distinta.

Se ha de mencionar que dichos métodos, SETy GET, han causado un debate entre defensores y detractores, debido a que se pierde seguridad de acceso a los atributos de nuestra clase en cuestión.

Tanto SET como GET son la forma de acceder a atributos de una clase, son métodos públicos, GET se va a encargar de mostrar un dato, es decir cuando lo que queremos es obtener dicho dato o valor de la variable o atributo, por el contrario SET se va a utilizar para modificar el valor del atributo, o mejor dicho gracias a éste método podemos asignar o reasignar un valor a una variable.

Para comprender mejor el uso de estos métodos se va a modificar los atributos de la clase Gato, haciendolos privados, con lo cual eso implica que se debe implementar los métodos públicos GET y SET, el motivo es que gracias a eso, desde una clase externa se puede acceder a los atributos privados haciendo uso o llamando a dichos métodos públicos.

```
public class Gato {

  /** ATRIBUTOS **/

  private String nombre;
  private double peso;
  private String color;

     /** MÉTODOS GET Y GET DE LOS ATRIBUTOS PRIVADOS**/

     public String getNombre() {
  return nombre;
  }
  public void setNombre(String nombre) {
  this.nombre = nombre;
  }

  public double getPeso() {
  return peso;
  }
  public void setPeso(double peso) {
  this.peso = peso;
     }

  public String getColor() {
  return color;
     }
  public void setColor(String color) {
  this.color = color;
  }

  /** MÉTODO imprimirPorConsola**/
```

```
    public void imprimirPorConsola() {
        System.out.println();
        System.out.println("nombre: " + this.nombre);
        System.out.println("color: " + this.color);
        System.out.println("peso: " + this.peso);

    }
}
```

Debemos de ser muy cuidadosos con la nomenclatura de los métodos GET y SET, los nombres de los métodos, empiezan set o get según sea el caso, seguido del nombre de la variable o atributo, con la particularidad de que la primera letra esté en mayúscula.

Por ejemplo si tenemos el atributo **color**, los métodos SET y GET se llamarán **setColor** y **getColor** respectivamente.

Cabe mencionar que el método set es de tipo void, es decir sin retorno, ya que solo se va a encargar de asignar o reasignar el valor del atributo que nos interesa, casi al terminar este subapartado explicaré que son las funciones de tipo void, y las de tipo retorno.

Continuando con los métodos SET y GET, y su importante utilidad a la hora de acceder a atributos privados de una clase, la cual en nuestro ejemplo es la clase Gato. Ahora se verá como la clase clasePrincipal va a utilizar los métodos SET y GET para acceder a los atributos privados:

```
  public class clasePrincipal {

    public static void main(String[] args) {
        Gato gato1 = new Gato();
        gato1.setNombre("Turco");
        gato1.setColor("amarillo");
        gato1.setPeso(1.5); /* peso en kilogramos */
        gato1.imprimirPorConsola();

        Gato gato2 = new Gato();
        gato1.setNombre("Tom");
        gato1.setColor("negro");
        gato1.setPeso(2); /* peso en kilogramos */
        gato2.imprimirPorConsola();
          }
}
```

Se puede apreciar como la clase principal, hace uso del método SET para asignar los valores de los atributos nombre, color y peso para cada uno de los dos objetos (gato1 y gato2) pertenecientes a la clase Gato, ahora lo que se hará, será crear un nuevo método en clasePrincipal para hacer uso del método GET:

```
   private static void imprimirPorConsola(Gato gato) {
     System.out.println();
     System.out.println("nombre: " + gato.getNombre());
     System.out.println("color: " + gato.getColor());
     System.out.println("peso: " + gato.getPeso());

}
```

Puesto que ahora la clasePrincipal tiene su propio método imprimirPorConsola, ya no hace falta llamar al método imprimirPorConsola del método Gato, con lo que la clase clasePrincipal quedaría así:

```
   public class clasePrincipal {

     public static void main(String[] args) {
         Gato gato1 = new Gato();
         gato1.setNombre("Turco");
         gato1.setColor("amarillo");
         gato1.setPeso(1.5); /* peso en kilogramos */

         Gato gato2 = new Gato();
         gato1.setNombre("Tom");
         gato1.setColor("negro");
         gato1.setPeso(2); /* peso en kilogramos */
     }
        private static void imprimirPorConsola(Gato gato) {
     System.out.println();
     System.out.println("nombre: " + gato.getNombre());
     System.out.println("color: " + gato.getColor());
     System.out.println("peso: " + gato.getPeso());
        }
   }
```

Después de que se ha explicado los métodos GET y SET (los cuales se consideran muy importantes, demás está decir que dichos métodos fueron muy útiles, en la aplicación Android que se ha desarrollado la cual es el corazón de este proyecto, el cual se detallará más adelante) ahora se procederá a explicar ya por terminar éste subapartado lo que son las funciones tipo **void** y las de **tipo con retorno** así como la palabra reservada **this.**

FUNCIONES DE TIPO VOID: Las funciones de tipo void o también llamadas sin retorno, son justamente eso, funciones que por lo general, no llevan la sentencia **return**, es decir no tienen que devolver ningún valor, sobre el cual tengamos que hacer otro cálculo u operación posterior, generalmente se va a utilizar éste tipo de funciones cuando se necesite imprimir un texto por pantalla o consola, o cuando se desee asignar un valor a una variable, y dicho valor posiblemente se sobrescriba con lo cual no nos interese conservarlo (razón por la cual no nos interesa que se devuelva dicho valor).

Para que esto quede más entendible se va a proponer un ejemplo de una conversación muy coloquial entre dos personas (quizás se esté pensando que esto no tiene nada que ver, pero lo cierto es que al autor de estas líneas, le ayudó mucho a comprender las funciones tipo void):

- **Persona A (programador):** Persona B, quiero que te desplaces hacia la izquierda.
- **Persona B (función o método):** Vale.
- **Persona A (programador):** ..... ¿Y? …
- **Persona B (función o método):** Y ¿qué?... no te entiendo, solo me pediste que me desplazara hacia la izquierda.
- **Persona A (programador):** Pero…no me dijiste si te desplazaste o no.
- **Persona B (función o método):** No fue eso lo que me pediste.

Aunque parece gracioso el ejemplo anterior, se puede entender perfectamente el significado de las funciones tipo void, en dicho ejemplo de la conversación se puede apreciar como la persona B o mejor dicho la función o método, se encarga exclusivamente de desplazarse hacia la izquierda, más no de confirmar o notificar si lo hizo o no.

Un ejemplo más formal sería el de calcular la suma de dos números e imprimir el resultado por pantalla:

```
public class prueba {
public static void main (String [] args){
sumar (3,5);
   }
public static void sumar (int a, int b){
int resultado;
resultado = a +b;
System.out.println("El resultado de la suma es " +resultado);
}
 }
```

Se puede apreciar como la función void sumar, se limita a calcular la suma y mostrarla por pantalla, si nos damos cuenta, la función sumar, no devuelve el resultado de la suma a la función que la llamó (la función main fue quien llamó a la función sumar).

Luego de explicar la función tipo void, ahora se explicará las funciones de tipo retorno.

FUNCIONES DE TIPO RETORNO: Las funciones de tipo retorno (usaran siempre la sentencia **return**) después de ejecutar su código van a devolver siempre un valor, ya que dicho valor será utilizado para posteriores operaciones o comparaciones según sea el caso.

Al igual que se hizo con las de tipo void, se va a poner un ejemplo de una conversación muy coloquial entre dos personas, con el fin de comprender el significado de las funciones con retorno:

- **Persona A (programador):** Persona B, quiero que te desplaces hacia la izquierda.
- **Persona B (función o método):** Vale.
- **Persona B (función o método):** Ya lo hice.
- **Persona A (programador):** Perfecto.

En esta conversación se puede ver no solamente que la persona B (función o método) cumple con su tarea encomendada, sino también que se lo comunica a la persona A (programador).

A continuación se muestra un ejemplo en java:

```
public class prueba {
public static void main (String [] args){
   int c;
   int d;
c = sumar(3,5);
   d = c*3;
   System.out.println("El resultado final es: " +d);
}
public static int sumar(int a, int b){
int resultado;
resultado = a +b;
   return resultado;
}
 }
```

Se puede ver que en éste programa, se va a calcular la suma de los números 3 y 8, para ello se llama a la función de tipo retorno sumar, el cual devolverá el valor de la suma a la función main, que fue quien la llamó, para posteriormente hacer otra operación (triplicar dicho valor).

NOTAS ACLARATORIAS: A lo largo de lo que se lleva explicando, se ha estado usando indistintamente, los términos método y función, lo cual técnicamente o formalmente no sería correcto, un método en java es un conjunto de instrucciones definidas dentro de una clase, que realizan una determinada tarea, a las que podemos llamar o invocar mediante un nombre.

Ahora bien, un método puede ser de 2 tipos, si el método no devuelve ningún valor diremos que es un procedimiento, o también conocido como de tipo void, en cambio si el método devuelve o retorna un valor diremos entonces que se trata de una función. El autor de estas líneas, nunca suele usar el término de procedimiento, ya que desde que empezó a programar, suele hablar de funciones con retorno o función sin retorno (tipo void), y no quería dejar éste subapartado sin aclarar los términos procedimiento y función.

PALABRA RESERVADA THIS: La palabra clave o reservada **this**, se usa cuando haya ambigüedad, entre un parámetro de un método y una variable (ambos tienen el mismo nombre).

A continuación vamos a ver un uso de this, con un ejemplo ya visto anteriormente:

```
public class Gato {

 /** ATRIBUTOS **/

 private String nombre;
 private double peso;
 private String color;

   /** MÉTODOS GET Y GET DE LOS ATRIBUTOS PRIVADOS**/

   public String getNombre() {
 return nombre;
 }
 public void setNombre(String nombre) {
 this.nombre = nombre;
 }

 public double getPeso() {
 return peso;
 }
 public void setPeso(double peso) {
 this.peso = peso;
   }

 public String getColor() {
 return color;
   }
 public void setColor(String color) {
 this.color = color;
 }

 /** MÉTODO imprimirPorConsola**/

 public void imprimirPorConsola() {
     System.out.println();
     System.out.println("nombre: " + this.nombre);
     System.out.println("color: " + this.color);
     System.out.println("peso: " + this.peso);

 }
}
```

Se puede ver que la clase Gato tiene tres atributos, también conocidos como variables o campos:

```
/** ATRIBUTOS **/

  private String nombre;
  private double peso;
  private String color;
```

Ahora bien, si nos fijamos en los métodos SET, vemos que dichos métodos tienen parámetros de igual nombre que los atributos de la clase Gato:

```
public void setNombre(String nombre) {
  this.nombre = nombre;
}

public void setPeso(double peso) {
  this.peso = peso;
}

public void setColor(String color) {
  this.color = color;
}
```

Lo que se hace con this.XXXXXX es evitar la ambigüedad, ya que con el uso de this estamos diciendo que accedemos al campo XXXXXX del objeto, es decir que this nos indica una referencia al objeto.

De igual manera ocurre en el siguiente método:

```
/** MÉTODO imprimirPorConsola**/

public void imprimirPorConsola() {
    System.out.println();
    System.out.println("nombre: " + this.nombre);
    System.out.println("color: " + this.color);
    System.out.println("peso: " + this.peso);

  }
```

De nuevo, con la palabra this, se accede al campo nombre, color y peso del objeto de la clase Gato, notemos que éste método imprimirPorConsola, no recibe ningún

parámetro, con lo cual no hay ambigüedad entre un campo (variable o atributo) y parámetro, por lo que el uso de this, aunque no está mal, no era necesario, pudiendo quedar el método de la siguiente manera:

```
/** MÉTODO imprimirPorConsola**/

public void imprimirPorConsola() {
    System.out.println();
    System.out.println("nombre: " + nombre);
    System.out.println("color: " + color);
    System.out.println("peso: " + peso);

  }
```

Se debe recordar siempre que en caso de que hubiese ambigüedad, el que predomina siempre es el campo, es decir el atributo o variable.

A continuación se seguirá ejemplificando los tres últimos conceptos que son: polimorfismo, clases abstractas e interfaces, se tiene que resaltar que todos los conceptos tratados en éste capítulo cinco, volverán a aparecer en el capítulo seis de programación en Android.

5.4.5 POLIMORFISMO

En el subapartado 5.3.3 se explicó que el polimorfismo (varias formas), es un concepto que va a estar muy ligado al concepto de herencia, en el que en resumidas cuentas, una variable de la clase padre, puede tomar la forma de cualquiera de sus clases hijas, esto se entenderá mejor con el siguiente ejemplo:

```
public class Persona{

    /** ATRIBUTOS **/
      public String nombre;
      public String apellido1;
      public String apellido2;
      public String sexo;
      public int edad;

    /** MÉTODOS PÚBLICOS**/
      public void imprimeSaludo() {
      System.out.println("Hola soy una persona ");
  }
```

```
    public void hablar() {
        System.out.println("La persona ahora habla");
    }
    public void caminar() {
        System.out.println("La persona ahora camina");
    }
      public void comer() {
        System.out.println("La persona ahora come");
    }
      public void dormir() {
        System.out.println("La persona ahora duerme");
    }
      public void despertar() {
        System.out.println("La persona ahora despierta");
    }
      public void estudiar() {
        System.out.println("La persona ahora estudia");
    }
    public void trabajar() {
        System.out.println("La persona ahora trabaja");
    }

}
```

Tenemos la clase Persona (ya visto anteriormente), que será la clase padre, del cual heredarán sus metodos y atributos sus clases hijas Programador, Futbolista y Profesor:

```
   public class Futbolista extends Persona {
         public String nombre_equipo;
         public String posicion_juego;


         public void imprimeSaludo() {
         System.out.println("Hola soy un buen futbolista ");
    }
}
```

```
   public class Programador extends Persona {
         public String sistema_operativo;
         public String marca_ordenador;


         public void imprimeSaludo() {
         System.out.println("Hola soy un programador ");
    }
}
```

```
   public class Profesor extends Persona {
```

```
        public String centro_estudio;
        public String asignatura;
        public String nombre;

        public void imprimeSaludo() {
         System.out.println("Hola soy un profesor ");
    }
}
```

Ahora tendremos una **clase principal**, la cual se llamará **pruebaPolimorfismo**, en la cual se verá la aplicación del polimorfismo:

```
  public class pruebaPolimorfismo {

    public static void main(String[] args) {
                Programador prog  = new Programador();
                prog.nombre = "Antonio";
                prog.edad = 27;

                Profesor profe = new Profesor();
                profe.nombre = "Carlos";
                profe.centro_estudio = "UPNA";

                Futbolista fut = new Futbolista();
                fut.nombre = "Sergio";
                fut.nombre_equipo = "REAL MADRID";

                Persona person = new Futbolista();
                person.imprimeSaludo();
                person = new Programador();
                person.imprimeSaludo();
        }
    }
```

Es a partir de aquí donde se aplica el uso del polimorfismo, se puede ver como un objeto de tipo Futbolista se va a almacenar en una variable de tipo Persona (realmente se conoce como variable contenedora), es decir son de distinta clase (Persona y Futbolista) y también un objeto de la clase Programador se va a almacenar en dicha variable contenedora, en resumidas cuentas podemos decir que esa variable de la clase padre toma varias formas de sus clases hijas, esto es el POLIMORFISMO.

5.4.6 CLASES ABSTRACTAS

Las clases abstractas, como ya se explicó hace varias páginas atrás, es una clase de la cual no se va a generar objeto alguno, es decir, es una clase que no se puede instanciar, pero sí es una clase que puede ser heredada, o lo que es lo mismo, sí puede tener clases hijas.

A continuación tenemos el prototipo de una clase abstracta:

```
public abstract class Principal{

  /**Método concreto implementado*/
  public void metodoConcreto(){
      ..............
      ..............
  }


   /**Método Abstracto (no implementado)*/
  public abstract void metodoAbstracto();

 }
```

Aquí podría haber varios métodos implementados de la clase abstracta, así como también métodos abstractos, pero con que haya al menos un método abstracto (es decir no implementado) de dicha clase abstracta es suficiente para que se defina como clase abstracta.

Nótese en la palabra reservada **abstract,** con lo cual se declara o define un método abstracto.

Por otra parte tenemos la clase que hereda de dicha clase abstracta:

```
class claseHija extends Principal{

  @Override
  public void metodoAbstracto() {
  /**Implementación definida
       por la clase concreta**/
  }

}
```

Esta clase que hereda de dicha clase abstracta (clase padre) tiene que implementar cada uno de los métodos que posea la clase abstracta.

La anotación **@Override**, nos indica que el método que viene a continuación va a ser implementado, modificado o sobrescrito.

5.4.7 INTERFACES

Las interfaces son como las clases abstractas, con la diferencia de que no implementan ningún método, es decir que todos sus métodos son abstractos, cabe

destacar que las clases pueden implementar tantas interfaces como quisieran, es por eso que el término **extends** no tiene sentido para esas clases en el campo de las interfaces, ya que se debe recordar que en Java no existe herencia múltiple, por el contrario sí que tiene sentido el término **implements**. Cabe resaltar que una clase puede implementar varias interfaces. Veamos un ejemplo de una interfaz:

```
   interface InterfacePrincipal {

     public void metodoAbstracto();

     public String otroMetodoAbstracto();

}
```

En las interfaces todos los métodos son abstractos, es por eso que no se aprecia la palabra reservada **abstract** en cada uno de los métodos, porque se da por entendido que todos los métodos de la interfaz no están implementados, y no hace falta ir añadiendo dicha palabra en cada uno de los métodos.

Ahora veamos una clase que hace uso de interfaces, o mejor dicho que las implementa:

```
   public class Principal implements InterfacePrincipal,segundaInterface,masInterface{

   public void metodoAbstracto() {
     /**Implementación definida por la clase Principal*/
     }

     public String otroMetodoAbstracto() {
     /**Implementación definida por la clase Principal*/
         return "retorno";
     }

     public void metodoAbstractoDesegundaInterface() {
     /**Implementación definida por la clase Principal*/
     }

     public void metodoAbstractoDemasInterface() {
     /**Implementación definida por la clase Principal*/
     }
}
```

En este ejemplo, vemos como la clase Principal, va implementar 3 interfaces y está en la obligación de implementar todos los métodos de dichas interfaces.

CAPÍTULO 6. PROGRAMACION EN ANDROID

Es este capítulo el más importante de todos los vistos hasta ahora, la programación en Android es el corazón de este proyecto, ya que en dicho proyecto se va a crear una aplicación para el móvil que reciba vía Bluetooth a través del módulo HC-05, los datos de las temperaturas enviados desde el arduino MEGA o UNO (cualquiera de ellos nos ha servido en este proyecto), en éste capítulo volverán a aparecer los conceptos vistos en Java, ya que las aplicaciones Android están escritas en Java, además veremos conceptos propios de Android, así como el uso del entorno de desarrollo Android Studio, el cual nos será de vital importancia a la hora de crear nuestras aplicaciones ya que nos facilita en gran medida dicha tarea, se ha de mencionar que antes de la aparición de Android Studio, se utilizaba otro entorno de desarrollo llamado Eclipse, pero ya está obsoleto, y en comparación con Android Studio se queda muy limitado, en cuanto a rendimiento, sencillez, atractivo visual, actualizaciones constantemente, y sinceramente los códigos escritos en Android, se ven más ordenados, vistosos y estructurados en Android Studio.

Como se mencionó en el apartado 2.2, Android fue diseñado principalmente para teléfonos móviles, pero actualmente éste sistema operativo abarca más dispositivos como relojes inteligentes, televisores e incluso automóviles.

Las aplicaciones Android están escritas en código Java, pero a diferencia de Java, no existe una máquina virtual Java como tal, en lugar de ello existe otra máquina conocida como Dalvik, ya que las aplicaciones Android están escritas en Java, en un principio (antes de ser compiladas) tienen la extensión **.java**, con lo cual si se compilan (con un compilador **Java**), tendrán una extensión **.class**, generándose así los bytecodes Java (de momento todo es Java), pero luego se da un paso muy importante, o mejor dicho un cambio trascendental, el cual es una transformación de bytecodes Java a bytecodes "Android", formalmente hablando son bytecodes Dalvik, que tendrán una extensión **.dex**, listos para ser ejecutados por la máquina Dalvik, generándose luego de la ejecución un archivo con extensión .**apk (**aplicación Android que el usuario tendrá en su dispositivo móvil). La máquina Dalvik fue creada específicamente para Android, y está optimizada para dispositivos móviles, los cuales está limitados en batería, memoria y microprocesador.

En éste gran capítulo se tratará además de los conceptos ya vistos en el tema de la programación orientada a objetos, se tratará el concepto de paquete, conceptos relacionados con la interfaz de usuario, también se estudiará la estructura de un proyecto Android, que no será más que una colección de carpetas (directorios) y archivos, que como se verá cada uno de ellos contendrá diferente tipo de información (por consiguiente también diferente tipo de formato), se recomienda solo centrarse en los directorios que contiene información tipo código de Java, permisos , recursos (y dentro del directorio recursos, principalmente en los subdirectorios layout y drawables), luego habrá otros archivos que será mejor no usarlos ni mucho menos modificarlos como los archivos Gradle, ya que lo único que hacen es causar confusión al programador. En éste capítulo se verá muy por encima el uso que debemos hacer en Android Studio para programar nuestras aplicaciones.

6.1 ANDROID STUDIO

Android Studio es el IDE, o mejor dicho el entorno de desarrollo integrado, oficial, lanzado por Google en 2014, el cual vamos a utilizar para poder desarrollar nuestras aplicaciones para el sistema operativo Android, definiéndolo de una forma muy simple diremos que es un editor de código con todas las herramientas de desarrollo necesarias.

Como ya se mencionó, las aplicaciones Android, están escritas en Java, con lo cual se debe tener instalado un software en nuestro equipo, para poder ejecutar código Java, y éste software se conoce cómo máquina virtual Java, Java Runtime Environment (JRE), o Virtual Java Machine.

Es muy probable que dicho software ya esté instalado en nuestros ordenadores, de hecho aunque no sea así, se suele avisar por medio de alertas o ventanas emergentes que dicho software nos hace falta para ejecutar o "correr" ciertas herramientas software, programas de diseño, juegos, reproducir videos de páginas web o cualquier otro contenido, etc.

Lo cierto es que tanto Android como Java están ligados, es imposible pensar que podemos programar en Android, sin Java, por ello, si no tenemos instalado la máquina virtual Java, debemos acceder a ésta dirección http://www.java.com/es/download/, debemos descargar e instalar el fichero que corresponda con nuestro sistema operativo.

Una vez instalado la máquina virtual Java, el siguiente paso es descargar e instalar el IDE Android Studio, a través de éste enlace https://developer.android.com/studio/index.html, cabe decir que el proceso de instalación de Android Studio es muy sencillo.

Figura 6-1 Ventana de arranque de la plataforma Android Studio.

Para tener un buen manejo de ésta plataforma, es conveniente entender ciertos conceptos, como las bibliotecas, API'S, SDK Manager, AVD, y demás herramientas que podemos encontrar en Android Studio.

BIBLIOTECA: Es un conjunto de funciones y procedimientos (una biblioteca puede ser desde tan básica hasta muy compleja) o mejor dicho subprogramas que trabajan en conjunto para una misma finalidad, y cuya existencia es la de ser utilizada por un programa, una biblioteca a diferencia de un programa ejecutable, no puede ser utilizada de forma autónoma, es más algunas bibliotecas pueden "servirse" o utilizar de otras, para funcionar.

Muchas veces, ya sea cuando se empieza o se lleva tiempo desarrollando software, surge la necesidad de utilizar ciertos recursos ("librerías") que ya hayan sido escritos (es justamente ello, lo más maravilloso de la programación), además el uso de bibliotecas o también conocidas como "librerías", les da una mejor estructura y organización a nuestro proyecto Android, cabe mencionar que para utilizar o mejor dicho para importar

bibliotecas a un proyecto, se usa la palabra reservada **import**, por poner un ejemplo, supongamos que necesitamos en un proyecto o programa Android que se esté desarrollando, la librería **android.view.View** (nótese la nomenclatura por medio de separación de puntos que tiene una librería), entonces lo que se tiene que hacer es escribir **import android.view.View**, consiguiéndose así que dicha librería esté disponible en dicho programa o proyecto Android, y de ésta manera poder utilizar todos los métodos (funciones y procedimientos) y clases que la librería incluye.

API: Se conoce como API, al conjunto de bibliotecas listas para ser usadas, cabe mencionar que API significa **interfaz de programación de aplicaciones** (del inglés Application Programming Interface), y es la forma alternativa en la que se clasifica e identifica la versión de Android que tengamos en nuestros dispositivos móviles. Actualmente el nivel máximo de API es el 23, que corresponde con la versión 6.0 del sistema operativo Android.

SDK MANAGER: De momento se tiene claro, que el entorno de desarrollo se llama Android Studio, el cual va a permitir escribir nuestros programas, pero a su vez dicho entorno de desarrollo va a contener diversas herramientas muy útiles entre ellos el SDK Manager, que significa gestor del kit de desarrollo de software (del inglés Software Development Kit), el SDK Manager se va encargar de gestionar las descargas de las API's de Android, que como ya se explicó son las diferentes bibliotecas (erróneamente conocidas como librerías) de las distintas versiones del sistema operativo Android.

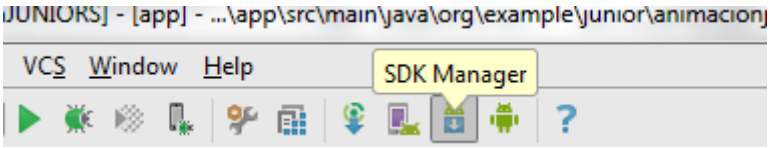


Figura 6-2 Simbolo de la herramienta SDK Manager en Android Studio.

AVD: Es otra herramienta de nuestro IDE, la cual va a permitir emular un dispositivo Android, o mejor dicho nos va a permitir crear dispositivos virtuales Android (del inglés Android Virtual Device), con unas cierta características, la cual podemos ajustarlas a nuestras necesidades, y lo más interesante es que podemos correr nuestras aplicaciones en dichos dispositivos virtuales.

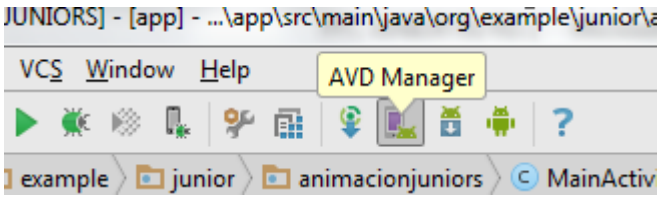


Figura 6-3 Simbolo de la herramienta AVD Manager en Android Studio.

6.2 COMPONENTES DE UNA APLICACION

Ahora se va a proceder a explicar unos conceptos que son esenciales para entender la programación en Android, ya que si bien es cierto, nuestro propósito es crear o mejor dicho desarrollar una aplicación Android, y para ello se tiene que saber cuáles son los componentes de una aplicación, en este apartado se estudiarán los componentes principales y éstos son: activity, view, layout, e intent, además son éstos los que nunca faltan en una aplicación Android de un cierto nivel de desarrollo, obviamente hay más componentes como: content provider (proveedor de contenido), service (servicio), broadcast receiver (receptor de anuncios) y fragment (fragmento).

6.2.1 ACTIVITY

Una activity o actividad va a ser la “cara principal” de la interfaz gráfica de usuario en Android (la interfaz gráfica de usuario, es aquel entorno visual que va permitir que el usuario interactúe con la aplicación), va a ser lo que comúnmente se va a llamar “pantalla de una aplicación”, también se define como aquel componente que va a ser el soporte para los distintos otros componentes visual, es decir en la actividad se va a poder introducir textos, botones, imágenes, definir una estructura de organización de todo lo que va a contener, en definitiva, vamos a poder elegir el orden y el posicionamiento de todos los demás componentes que van a estar en nuestra actividad (esta organización se consigue gracias al denominado layout, el cual será explicado más adelante), se reitera, que debemos quedarnos con la idea de que una actividad es la pantalla de nuestra aplicación, por consiguiente es lógico pensar que una aplicación va a tener muchas actividades, ya que cuando estamos navegando por una aplicación, estamos viendo los distintos contenidos, por ejemplo pensemos en la aplicación YouTube , en la cual vamos a tener una lista de videos disponibles, que dependiendo del video que vayamos a visualizar se abrirá una u otra nueva actividad.

6.2.2 VISTA

Una vista o view en inglés, va a ser esos componentes que se coloca en la actividad, como pueden ser: botones, cuadro de textos, imágenes, gestos en pantalla (conocidos como gestures en Android), estrellas de valoración (las típicas que aparecen cuando vamos a valorar una aplicación), scroll (vista de desplazamiento) etc. Es de vital importancia mencionar que toda la interfaz gráfica de usuario (IGU), se basa en una jerarquía de clases descendientes de la clase View (clases subclases o clases hijas, no se deben olvidar que dichos términos ya se vió en el capítulo anterior), con lo cual se puede asegurar de que una vista es el elemento base de la IGU.

6.2.3 LAYOUT

Se puede definir a un layout como un elemento no visual, que será el encargado de controlar la distribución, posicionamiento, organización y dimensionamiento de las vistas que están en su interior, con lo cual se puede decir que un layout es un contenedor de vistas.

Se va a aprovechar este subapartado, para mencionar que cuando los programadores diseñan la interfaz gráfica de usuario (IGU), son conscientes que dicha interfaz cobra cada día más importancia en el desarrollo de una aplicación, es por eso que se debe procurar que la IGU, sea lo menos compleja posible para el usuario.

Algo curioso en Android es que la interfaz gráfica de usuario (IGU), no está codificada en Java, por el contrario se utiliza para el diseño, un lenguaje de etiquetas conocido como XML (Extensible Markup Language o lenguaje de marcado extensible, en español), dicho esto, conviene explicar que XML, no es un lenguaje de programación, propiamente dicho, sino que es un lenguaje de marcado o etiquetas, similar a HTML, con XML se pretende separar en dos partes el proyecto Android que estemos realizando (lo cual es menos complejo y más fácil de desarrollar la aplicación), por una parte la lógica de aplicación y por otra, la parte del diseño.

En Android, hay varios tipos de Layout, los cuales son: LinearLayout, TableLayout, RelativeLayout, AbsoluteLayout y FrameLayout, a continuación se describirán dichos layout conjuntamente con sus ficheros XML.

LINEAR_LAYOUT: En éste tipo de Layout, las vistas contenidas en su interior, van a ir colocadas, linealmente, una detrás de otra (horizontalmente o verticalmente).

Podemos ver en ésta imagen, como los elementos (vistas) se distribuyen uno detrás de otro, en éste caso verticalmente

Figura 6-4 Ejemplo de LinearLayout.

A continuación, se muestra el fichero XML:

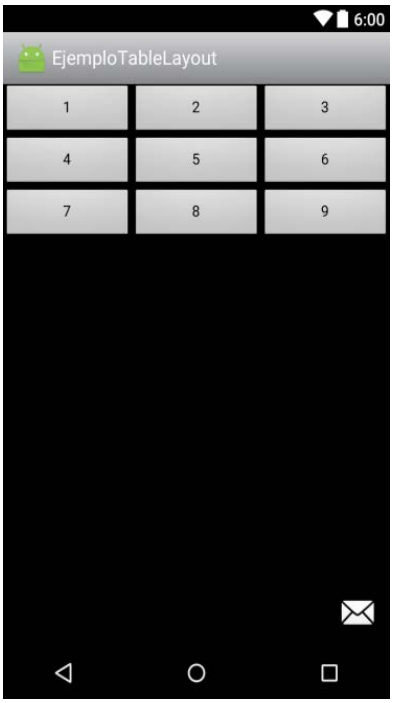
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    tools:context=".MainActivity">
    <TextView android:text="JUEGO JR"
        android:gravity="center"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
    <Button android:id="@+id/Button01"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Jugar"/>
    <Button android:id="@+id/Button02"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Configurar"/>
    <Button android:id="@+id/Button03"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Acerca de" />
    <Button android:id="@+id/Button04"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
        android:text="Salir"/>
</LinearLayout>
```

Realmente resulta muy intuitivo entender el fichero XML, en primer lugar se nos indica la versión de XML, y el tipo de codificación de caracteres utilizado para éste fichero, en segundo lugar se introduce un elemento de tipo LinearLayout, que como ya se explicó su función es la de contener vistas (elementos de tipo view), las cuales están alineadas verticalmente, dicho LinearLayout, en éste caso tiene siete atributos, los dos primeros definen espacios de nombres, conocidos también como declaraciones de espacios de nombres en XML, el tercer atributo como se puede intuir nos define la orientación , el cual en éste caso es vertical (podría ser también horizontal), el cuarto y quinto atributo nos definen la anchura y altura del layout respectivamente, los cuales tienen el valor de **match_parent**, lo cual quiere decir que tanto la anchura como la altura de dicho layout van ocupar todo el espacio máximo posible en pantalla. En el sexto atributo se encuentra **gravity**, el cual como su propio nombre indica va a definir la propiedad de gravedad del layout, lo cual nos da la idea de cómo están alineadas el conjunto de vistas, más no el contenido de ellas, en éste caso dicho atributo tiene el valor **center**, que significa que el conjunto de vistas se alinean por el centro de la pantalla (actividad), o que están al medio. El séptimo y último atributo (y el más importante) indica la actividad asociada a este layout.

En el interior del LinearLayout, tenemos a un cuadro de texto (también llamada etiqueta de texto) y cuatro botones, de los cuales viendo sus atributos, se va a explicar el significado de **wrap_content**, que quiere decir que la vista se ajustará a su contenido, es decir que la vista tiene que ser tal, que “cubra” lo que lleva en su interior, volviendo a las vistas de los botones vemos que la altura de ellos tienen el valor wrap_content, lo cual quiere decir que las alturas de los botones se ajustarán al texto contenido.

TABLE_LAYOUT: En este tipo de Layout, los elementos contenidos en su interior (vistas), estarán distribuidas de forma tabular.



Podemos ver en ésta imagen, como las vistas (en este caso botones) del Layout están distribuidas en forma de tabla.

Figura 6-5 Ejemplo de TableLayout.

Este tipo de layout, es de los pocos usados, ya que está diseñado para usos concretos, como el que se puede ver en la imagen superior.

A continuación, se muestra el fichero XML:

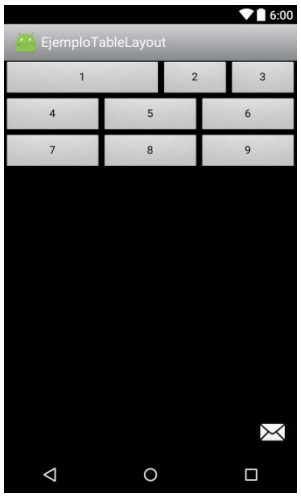
```
   <?xml version="1.0" encoding="utf-8"?>
<TableLayout
       xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#FF000000"
    tools:context=".MainActivity"
    android:orientation="vertical">
    <TableRow
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="1"
            android:layout_column="0"
            android:layout_weight="1" />
          <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
            android:text="2"
            android:layout_column="1"
            android:layout_weight="1" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="3"
            android:layout_column="2"
            android:layout_weight="1"/>
    </TableRow>

    <TableRow
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="4"
            android:layout_column="0"
            android:layout_weight="1" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="5"
            android:layout_column="1"
            android:layout_weight="1" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="6"
            android:layout_column="2"
            android:layout_weight="1"/>
    </TableRow>


      <TableRow
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="7"
            android:layout_column="0"
            android:layout_weight="1" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="8"
            android:layout_column="1"
            android:layout_weight="1" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="9"
            android:layout_column="2"
            android:layout_weight="1" />
    </TableRow>
</TableLayout>
```

Segun podemos ver, ahora tenemos una etiqueta correspondiente al Layout TableLayout, el cual tiene una serie de atributos, los cuales se ha explicado

anteriormente, salvo un término nuevo, el cual es el atributo **background** (color de fondo de la pantalla) que en este caso tiene el valor negro , cabe mencionar que los valores se indican en numeración hexadecimal, en formato ARGB, (alfa, rojo, verde y azul).

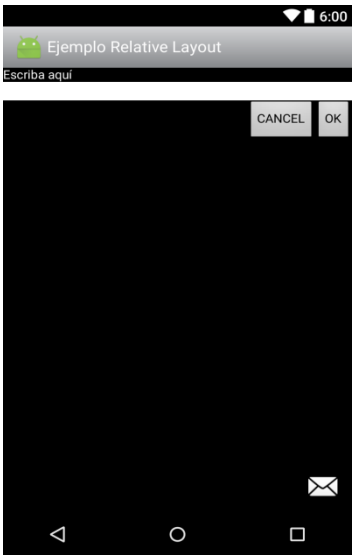
Se puede apreciar también una etiqueta llamada TableRow, cada vez que queramos insertar una nueva línea, en el ejemplo podemos ver como en cada TableRow tenemos tres botones, los cuales además de los atributos ya vistos anteriormente, tienen dos atributos los cuales son **layout_column** (el cual hace referencia al número de columna en el que se encuentra dicho botón) y **layout_weight** (el cual nos indica el peso que tiene dicha vista, es decir define quien va a tener mas "derecho" en ocupar mas espacio de pantalla o dicho de otra forma mide el nivel de importancia, que se traduce en más o menos espacio ocupado en la pantalla, de forma proporcional al valor asignado), podemos ver que los botones tienen el valor de 1 para la propiedad o atributo layout_weight, esto quiere decir que los tres botones de cada TableRow se reparten el espacio asignado equitativamente, pero si por ejemplo, uno de ellos tuviese el valor 3 en la propiedad layout_weight, entonces dicho botón ocuparía considerablemente más espacio que los otros dos botones



Podemos ver, en esta imagen el efecto práctico del uso del atributo **layout_weight** con un valor de 3, para el botón con el texto 1.

Figura 6-6 Ejemplo de TableLayout, aplicación de layout_weight.

RELATIVE_LAYOUT: En este tipo de Layout, las vistas van a ser distribuidas de forma relativa, es decir respecto a otras vistas, o respecto al layout que los contiene, cabe mencionar para las vistas , que el layout que las contiene, es el padre.



En este layout, tenemos una etiqueta de texto, una caja de edición de texto y dos botones, los cuales están posicionados de forma relativa, siempre respecto a otras vistas o del layout mismo, en el fichero XML, se podrá ver en más detalle dicha distribución.

Figura 6-7 Ejemplo de RelativeLayout.

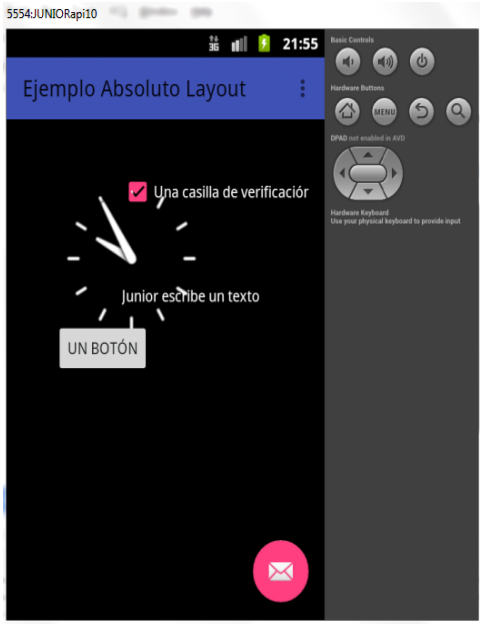
A continuación vamos a ver el fichero XML:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
       xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#FF000000"
      tools:context=".MainActivity">

    <TextView
        android:text="@string/texto"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textView"
        android:textColor="#FFFFFFFF"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText"
        android:textColor="#FF000000"
```

```
        android:background="#FFFFFFFF"
        android:layout_below="@+id/textView"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button"
        android:layout_below="@+id/editText"
        android:text="@string/okey"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cancelar"
        android:id="@+id/button2"
        android:layout_below="@+id/editText"
        android:layout_toLeftOf="@+id/button"
        android:layout_toStartOf="@+id/button" />

</RelativeLayout>
```

Analizando el fichero anterior, podemos ver la etiqueta de RelativeLayout, el cual tiene unos atributos, los cuales ya han sido explicados en otros tipos de Layout, dicho RelativeLayout, va a contener cuatro vistas, las cuales, además de los atributos ya vistos, vemos que tenemos otros atributos como identificador (**id**), el cual es usado por las demás vistas, como referencia para su posicionamiento. Veamos por ejemplo, el último elemento del fichero XML (la vista **Button**), el cual está identificado como **button2** (gracias a su atributo id, con valor **@+id/button2**), se encuentra debajo de la vista caja de edición de texto, identificada como **editText** (gracias a la propiedad **layout_below**), a la izquierda del otro botón (**layout_toLeftOf**), identificado como **button** y al empezar de dicha vista (**layout_toStartOf**).

ABSOLUTE_LAYOUT: Este layout va a posicionar sus vistas de forma absoluta, es decir que se va a poder indicar las coordenadas (x,y), donde vamos a querer que se visualice cada elemento (vista), de hecho esta forma de trabajar es mala, ya que la aplicación que estemos diseñando tiene que visualizarse correctamente en dispositivos con cualquier tamaño de pantalla, de hecho este tipo de layout ha sido marcado como obsoleto por Android.



Trabajar con AbsoluteLayout es muy incómodo, para cada vista que queramos insertar tenemos que especificar la coordenada cartesiana (x,y) de su colocación. En ésta imagen se puede ver también que dicho Layout ha sido ejecutado en un dispositivo virtual Android (AVD).

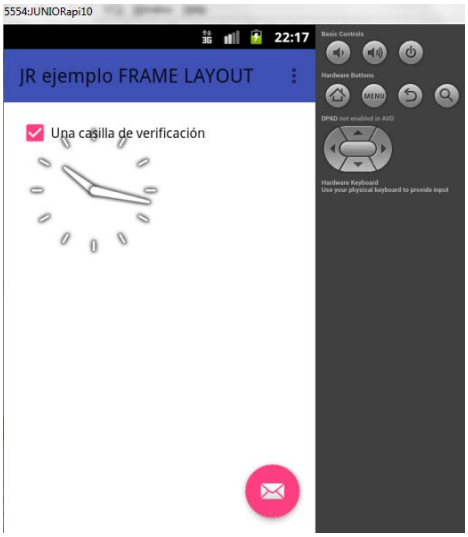
Figura 6-8 Ejemplo de AbsoluteLayout.

A continuación vamos a ver el fichero XML de este layout, en el cual podemos apreciar que para cada vista se especifica la coordenada absoluta en forma de píxeles (px):

```
   <?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
       xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#FF000000"
    tools:context=".MainActivity">
    <AnalogClock
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_x="50px"
        android:layout_y="50px" />
    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Una casilla de verificación"
        android:textColor="#FFFFFFFF"
        android:layout_x="150px"
        android:layout_y="50px"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Un botón"
        android:layout_x="50px"
        android:layout_y="250px"/>
    <TextView
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Junior escribe un texto"
        android:textColor="#FFFFFFFF"
        android:layout_x="150px"
        android:layout_y="200px"/>
</AbsoluteLayout>
```

FRAME_LAYOUT: Vamos a usar éste layout, cuando queramos que varias vistas ocupen un mismo lugar, es en éste caso, cuando hablar de distribución espacial no tiene sentido.



En esta imagen vemos como las dos vistas están superpuestas, no obstante ambas se llegan a ver con cierta claridad.

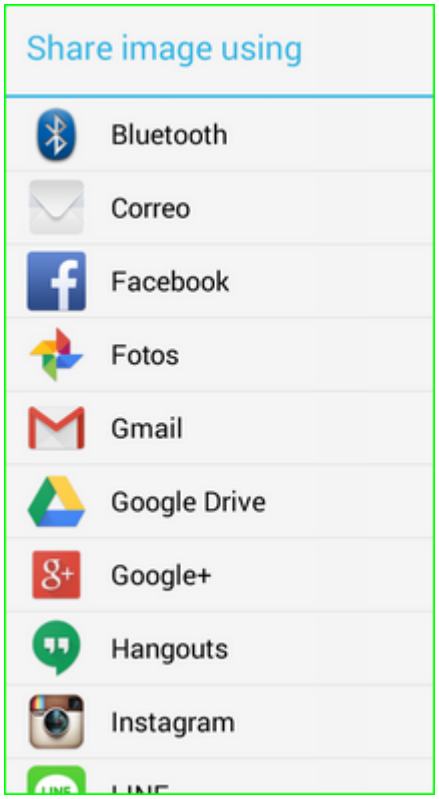
Figura 6-9 Ejemplo de FrameLayout.

A continuación vamos a ver el fichero XML:

```
   <?xml version="1.0" encoding="utf-8"?>
<FrameLayout
       xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <AnalogClock
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Una casilla de verificación"
        android:textColor="#FF000000" />
</FrameLayout>
```

6.2.4 INTENT

Un **intent** (o intención), es el componente que se puede definir como la "intención" o voluntad de realizar alguna acción, por ejemplo cuando desde nuestro dispositivo queramos enviar un mensaje, realizar una llamada, visualizar una página web o compartir un archivo, el componente i**ntent** se va a encargar de lanzar una serie de opciones para realizar dicha acción, pero los usos del intent van más allá, como por ejemplo, intercambiar información entre actividades (por lo general una aplicación va a tener más de una actividad), o simplemente también será la encargada de lanzar una nueva actividad, hemos de saber que muchos intents van a ser lanzados por el propio sistema Android, como por ejemplo: batería baja o llamada entrante, etc.



Podemos ver en esta imagen, como el Intent, nos ofrece una serie de aplicaciones opcionales, para realizar la acción de compartir una fotografía.

Figura 6-10 Uso del componente Intent.

6.3 CREACIÓN DE UN PRIMER PROGRAMA ANDROID

Después de haberse explicado los componentes de una aplicación Android, se va a entrar más en detalle sobre cómo se creará nuestra futura aplicación, y para ello el entorno de desarrollo Android Studio, nos va a facilitar dicha tarea, y esto se va a notar cuando empecemos a trabajar con la gran cantidad de ficheros que tiene Android.

A modo de pequeño tutorial se describirán los pasos para la creación de un programa en Android.

Lo primero que se tiene que hacer es iniciar Android Studio, para lo cual vamos a proceder como si se tratara de cualquier programa:

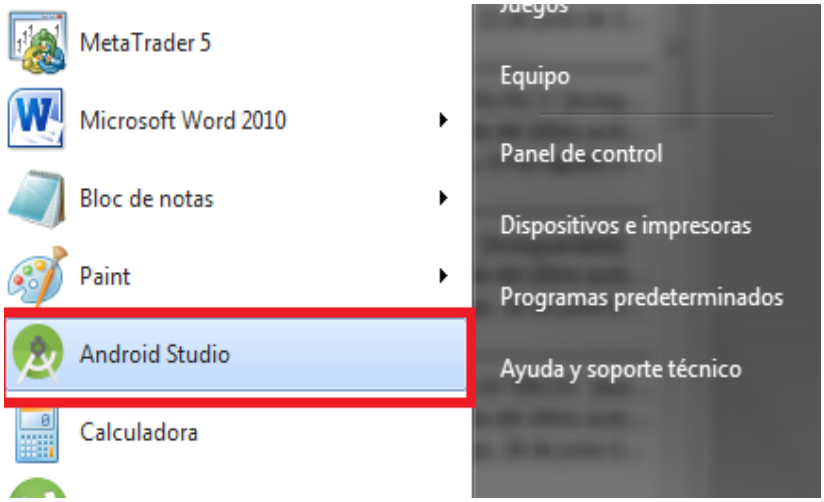


Figura 6-11 Localización de la plataforma Android Studio e inicio del mismo.

Luego se nos abrirá la siguiente ventana, en cuya parte izquierda estarán los programas (proyectos) que hayamos realizado recientemente, y en la parte derecha, habrá una serie de opciones, de las cuales deberemos hacer click en la primera, ya que lo que nos interesa es comenzar un nuevo programa (proyecto):

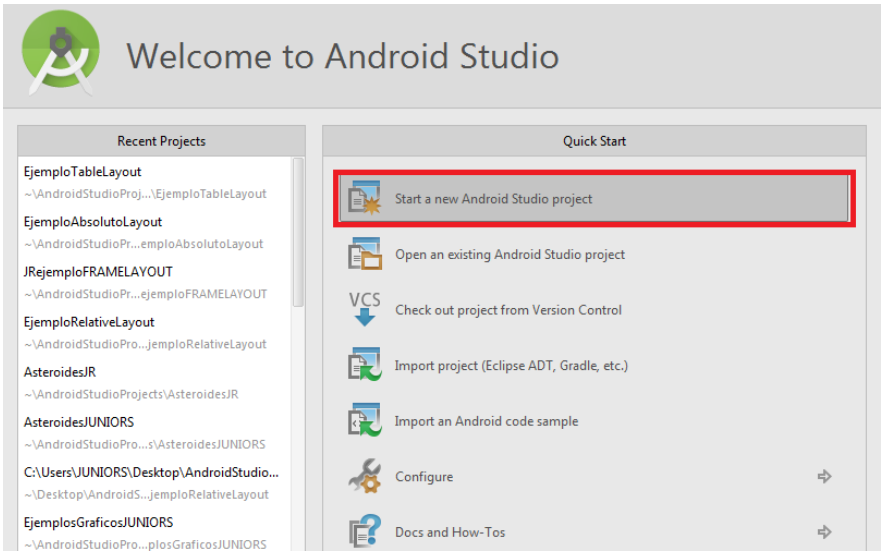


Figura 6-12 Ventana de bienvenida de Android Studio.

Una vez realizado el paso anterior, se nos abrirá el asistente de creacion de proyectos Android, en el cual tenemos que rellenar tres campos de datos, el primero de ellos es el nombre de la aplicación, el segundo es el dominio de la compañía, que va a ser un dominio web, utilizado por nosotros o por una empresa, para crear el paquete que contendrá nuestras clases java, tal como se puede apreciar en la linea siguiente del dominio de la compañía, el nombre del paquete se crea inviriendo los campos del dominio de la compañía y añadiendo el nombre de la aplicación. Se tiene que confesar que para un programador acostumbrado en programar en Java, indicar el nombre del paquete de esta forma, resulta un tanto extraño

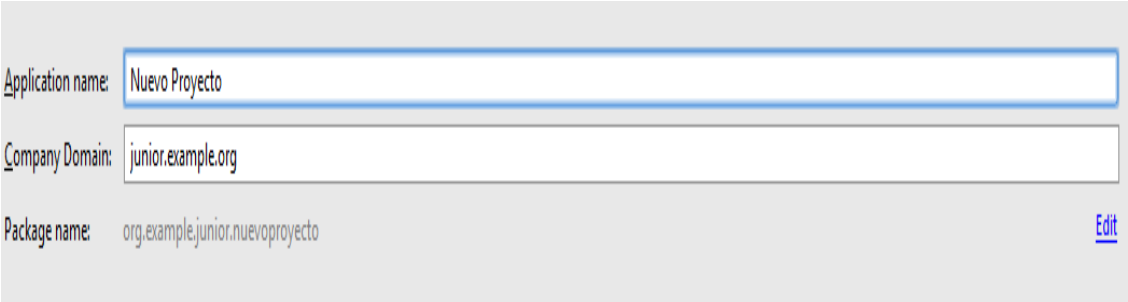


Figura 6-13 Los campos más importantes en el asistencia de creación de proyectos de Android Studio.

Y el último campo es la localización del poyecto, la cual nos va a permitir configurar la ruta donde se almacenará todos los ficheros de nuestro proyecto Android.

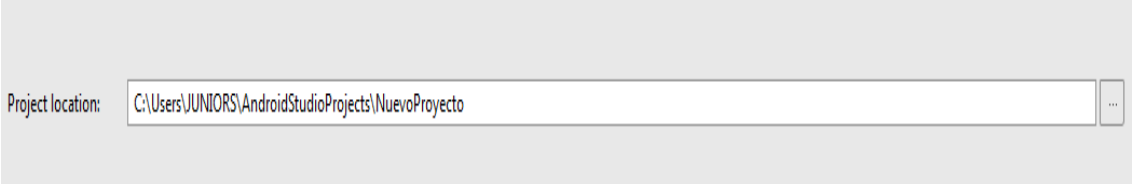


Figura 6-14 Campo de localización del proyecto de Android Studio.

Luego de haber rellenado los tres campos de datos anteriores, hacemos click en Next:

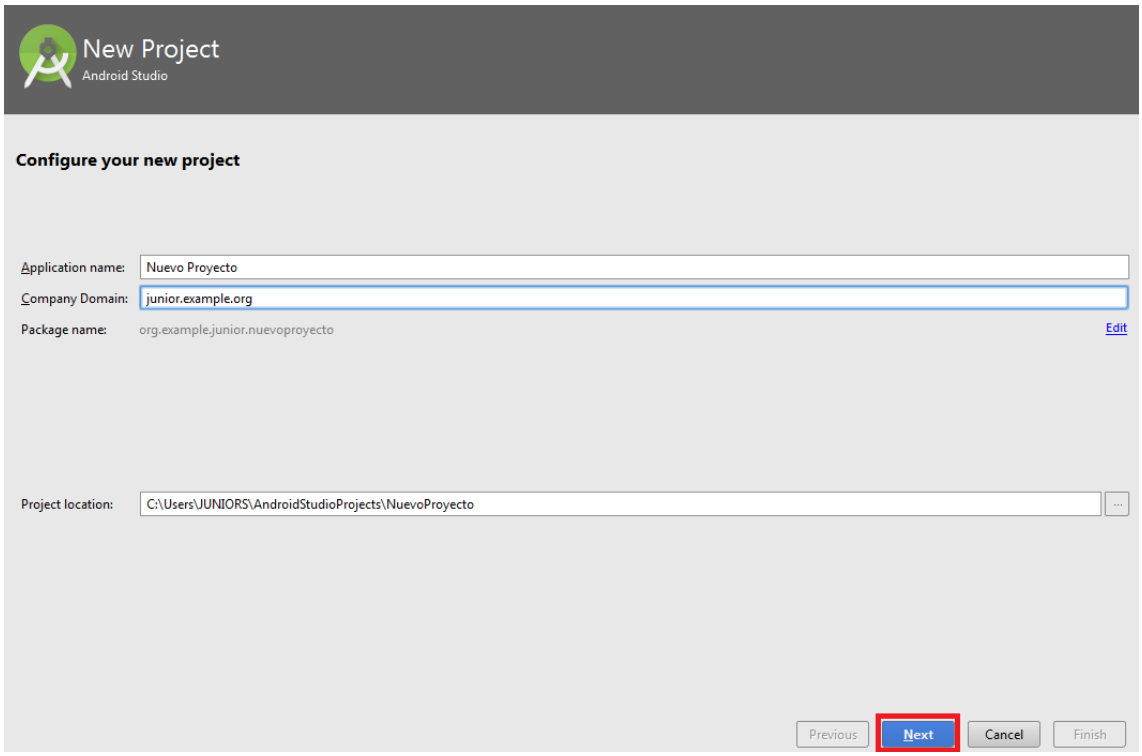


Figura 6-15 Asistente de creación de proyectos de Android Studio con todos los campos.

Luego se nos abrirá la siguiente ventana (imagen inferior), en la que tenemos que elegir dos cosas, la primera es el tipo de dispositivo sobre el cual queremos que se ejecute nuestra aplicación, en nuestro caso elegimos la primera opción, teléfono y tablet. La segunda es elegir el nivel minimo de API que requiere nuestra aplicación, que como ya anteriormente se explicó, el término API nos va a especificar con que tipo de version del sistema operativo Android, estamos trabajando, y aquí viene un tema muy importante, ya que los moviles con una versión anterior al especificado por nosotros, no podrán ni tan siquiera instalar nuestra aplicación. Es por ello que se aconseja elegir valores pequeños de API, para que nuestra aplicación pueda cubrir la mayor cantidad de dispositivos Android posible, se recomienda elegir un nivel de API 10, pero también tenemos que ser cuidadosos de no elegir valores tan pequeños ya que eso impediría poder utilizar las mejoras futuras disponibles para API's de mayores niveles, por poner un ejemplo; a partir del nivel de API 11 (o lo que es lo mismo versión 3.0 de Android) las mejoras de graficos de 2D y sobre todo en 3D, son realmente impresionantes, cosa que con API's de nivel inferior no existian, otro ejemplo seria nombrar la novedad introducida a partir del API 17 en la cual existe la posibilidad de conectar nuestro

smartphone Android con la TVHD, mediante wifi, lo que se conoce como Miracast, en resumen lo que buscamos siempre va a ser un compromiso, entre abarcar la mayor cantidad de dispositivos donde nuestra aplicación pueda correr y ser conscientes que ciertas novedades, particularidades o mejoras de software sólo están disponibles a partir de un nivel de API determinado.

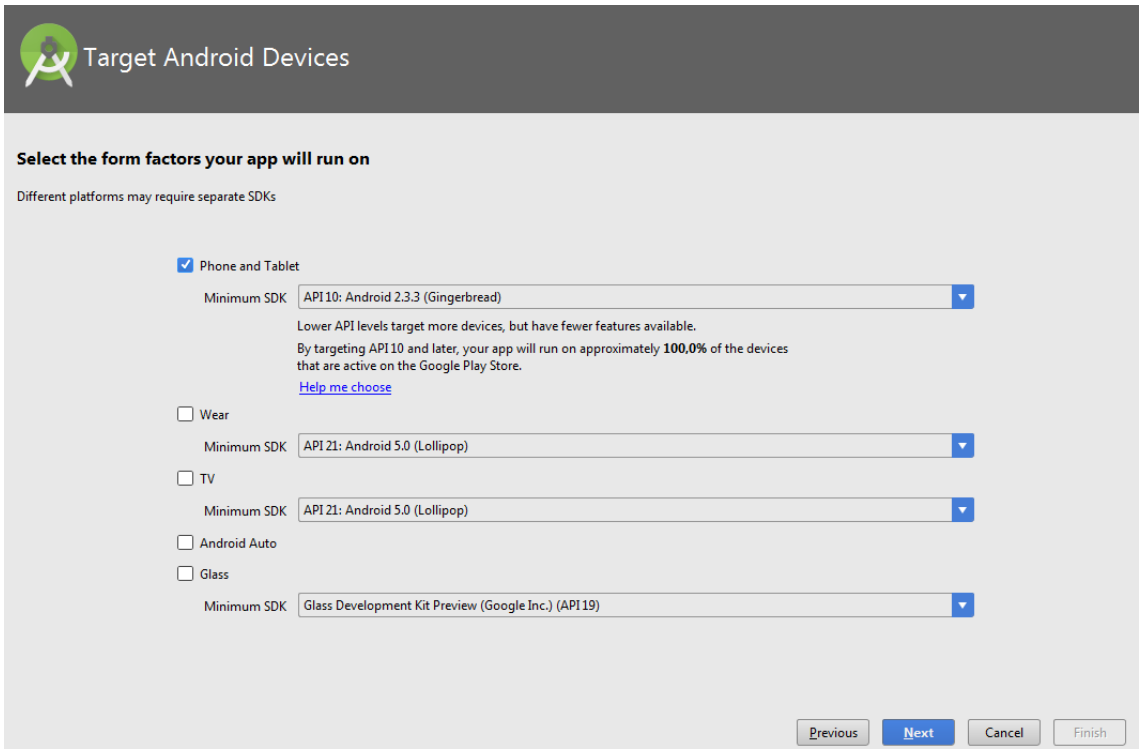


Figura 6-16 Elección del disposivo Android y API donde se ejecutará la aplicación.

En esa misma ventana (imagen superior), vamos a ir seleccionando diversas API's y a modo que aumentando el nivel de API, se aprecia como el porcentaje de dispositivos activos en la Play Store de Android en los cuales nuestra aplicación puede correr (ejecutar) disminuye.

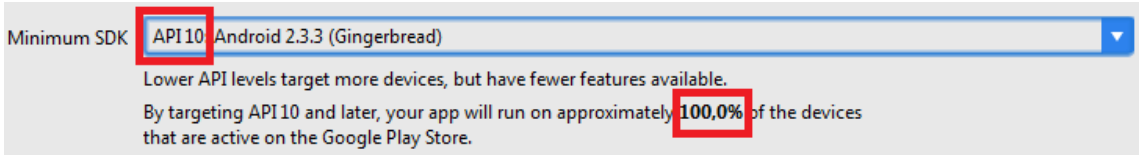
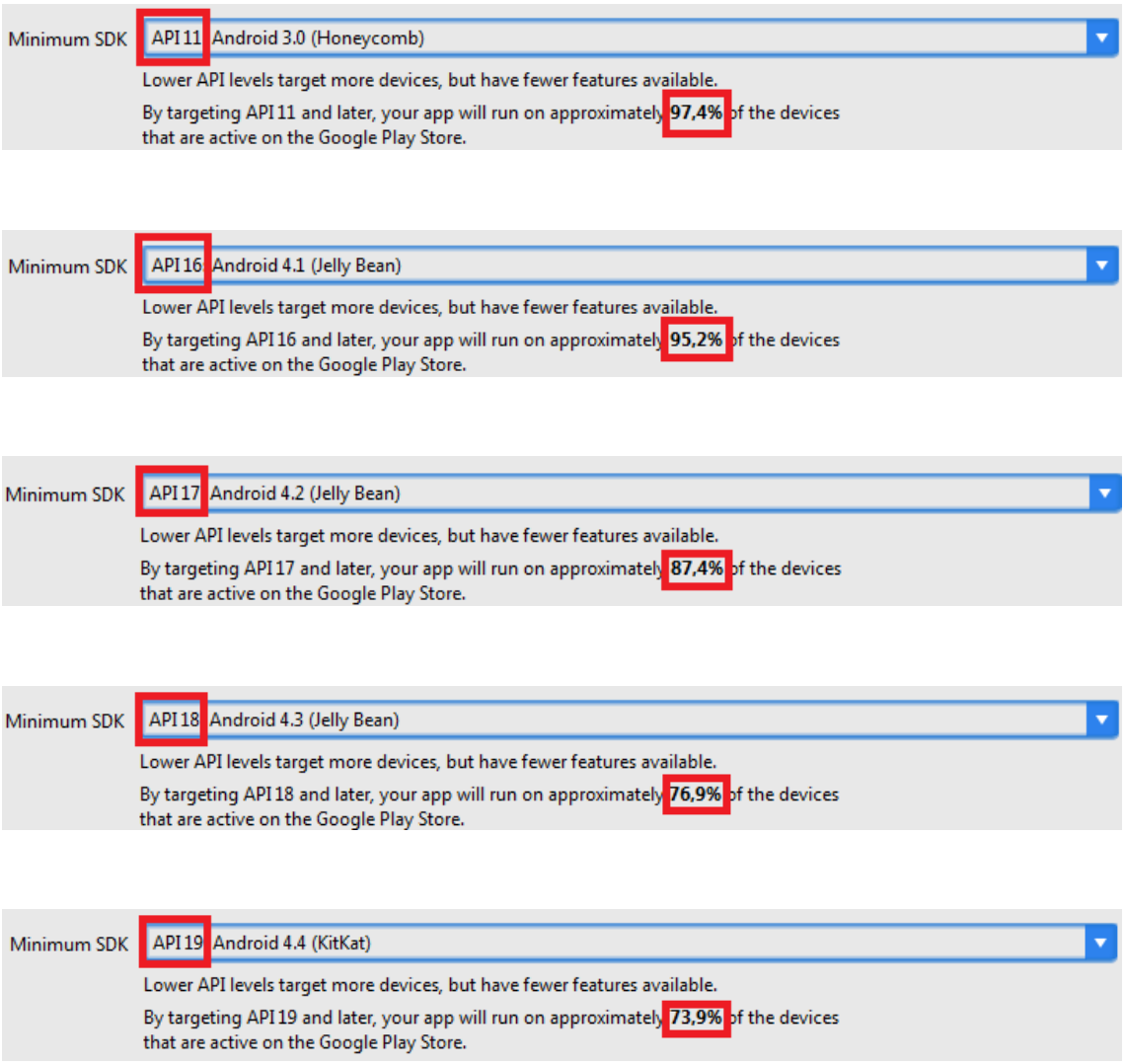
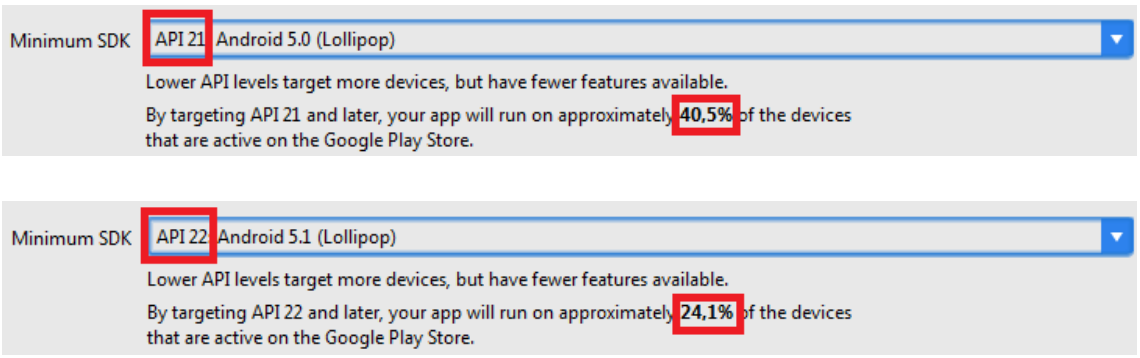


Figura 6-17 Relación entre el nivel de API y el porcentaje de dispositivos activos en la Play Store.



Figuras 6-18 Imágenes donde se puede seguir apreciando la relación entre el nivel de API y el porcentaje de dispositivos activos en la Play Store.

Es a partir del API 21 donde se aprecia una reducción importante del porcentaje de los dispositivos en los cuales podría correr nuestra aplicación.



Figuras 6-19 Reducción importante del porcentaje de dispositivos activos en la Play Store.

Y finalmente si usamos la última versión de Android, la API 23, esto es lo que ocurre (el porcentaje de mercado es tan bajo):

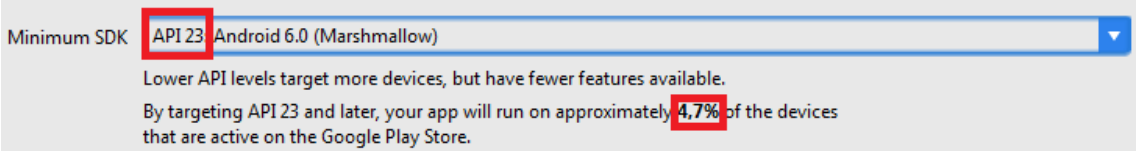


Figura 6-20 Reducción tan baja del porcentaje de dispositivos activos en la Play Store, cuando elegimos el nivel de API máximo.

Una vez, que hayamos elegido el tipo de dispositivo y el API minimo, damos click en Next:

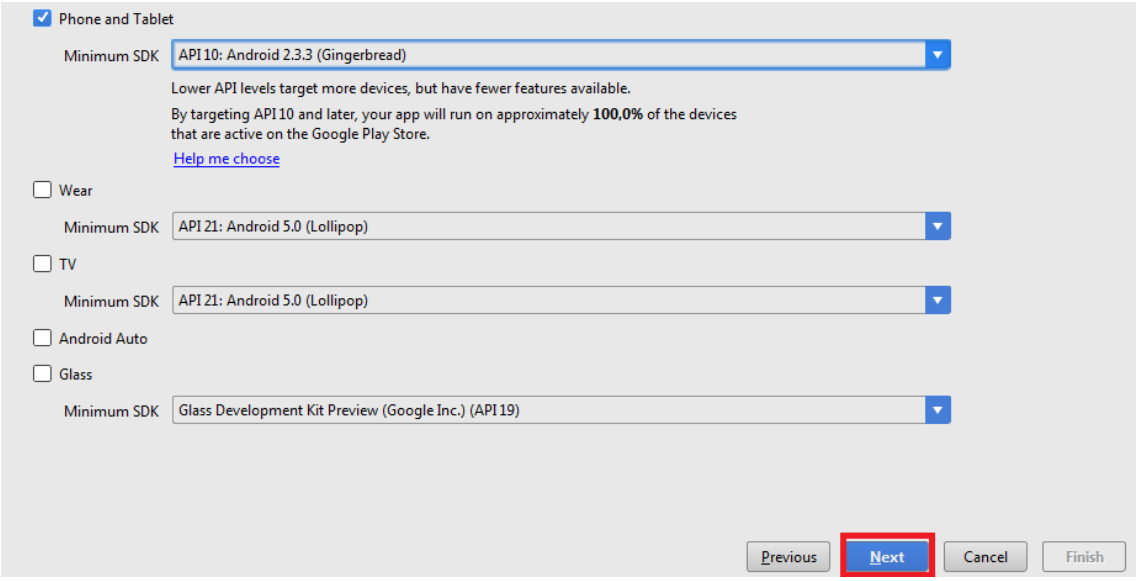


Figura 6-21 Elección del nivel de API y tipo de dispositivo para posteriormente clickear en Next.

Después de pulsar Next, se nos abrirá la siguiente ventana (imagen inferior), en la cual tenemos que elegir el tipo de actividad principal de la aplicación, (como ya se explicó en el subapartado 4.2.1, una actividad también es conocida como pantalla de la aplicación), en dicha ventana, tendremos una serie de actividades para elegir, de las cuales elegiremos la opción Empty Activity (actividad vacía).

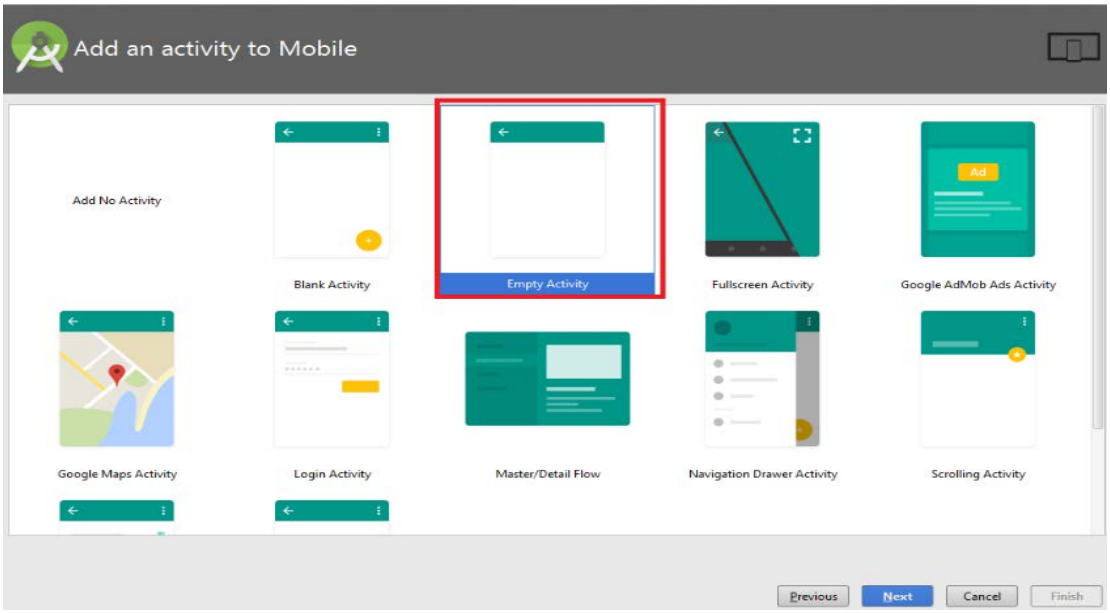


Figura 6-22 Elección de un actividad vacía (Empty Activity).

Elegimos la Empty Activity por ser la más sencilla de las actividades, y en el siguiente paso vamos a indicar los nombres de los elementos asociados a esta actividad que acabamos de elegir, como se puede ver en la siguiente imagen:

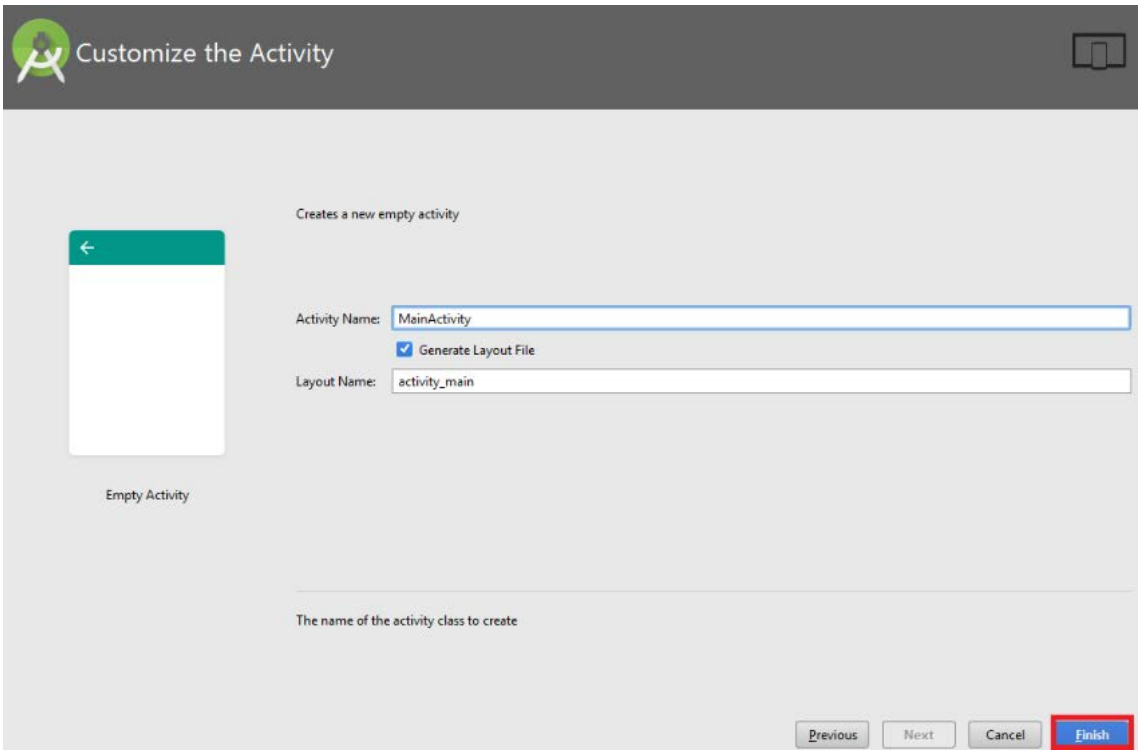


Figura 6-23 Particularización de la actividad.

En primer lugar indicamos el nombre de la actividad (o mejor dicho, el nombre de su clase java asociada) y en segundo lugar, el nombre del layout asociado a dicha actividad, dichos valores vienen por defecto (podremos modificarlos si queremos), los dejamos tal cual y pulsamos en finish para crear el proyecto:

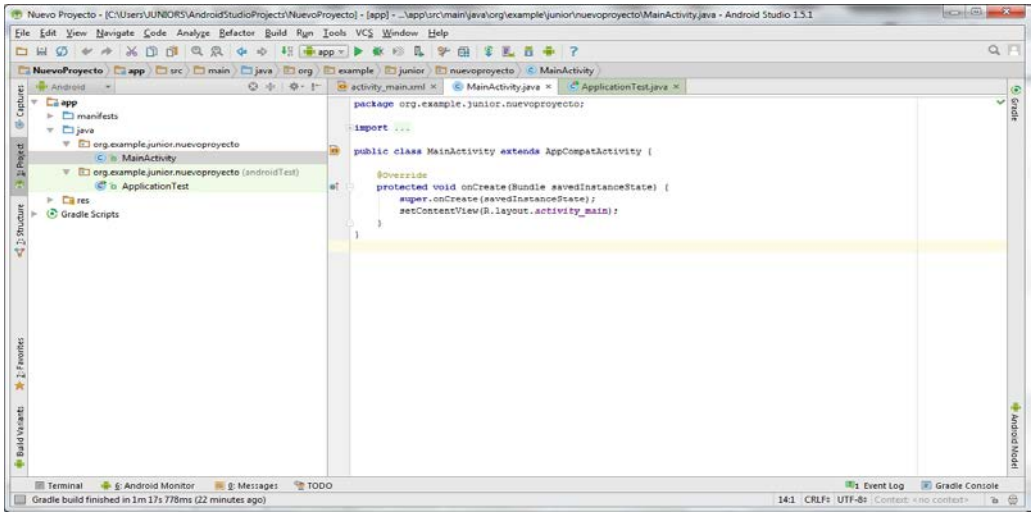


Figura 6-24 Ventana de un proyecto Android.

En la parte izquierda del proyecto, tenemos lo que se conoce como explorador del proyecto, va a ser el corazon de nuestro proyecto, todas las carpetas y ficheros de nuestro proyecto se van a accedar desde él:

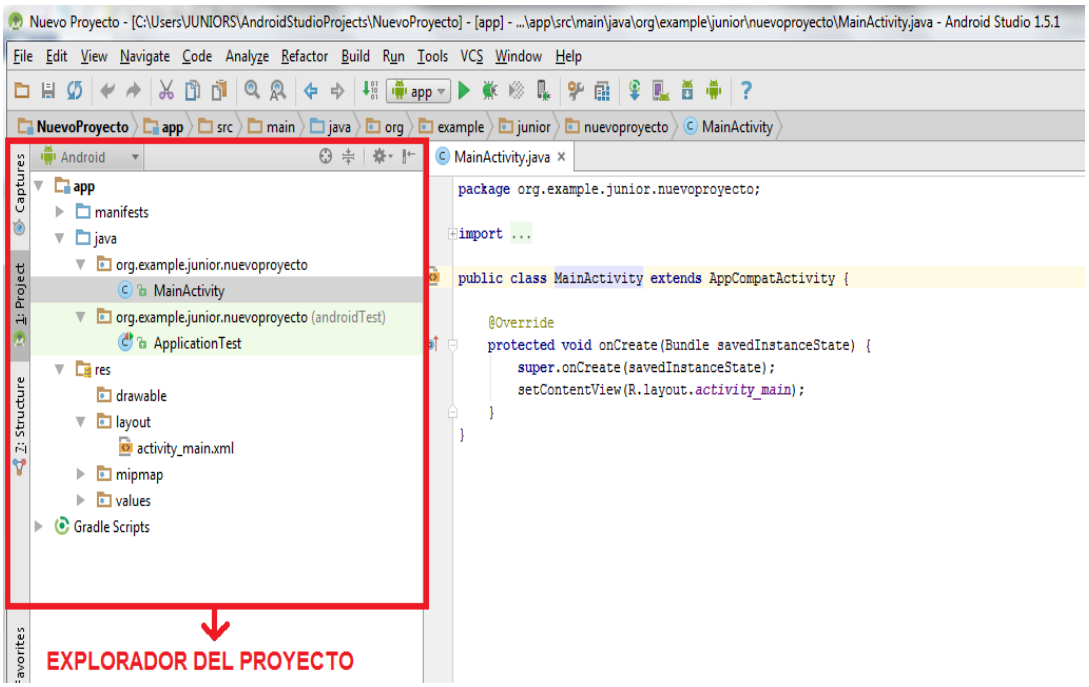


*Figura 6-25 Estructura del **explorador del proyecto**.*

Se puede apreciar como dicho **explorador del proyecto**, va a definir toda la estructura de nuestro proyecto, y los elementos indispensables que debe contener dicho proyecto.

Ahora se comentará un poco sobre la actividad, o coloquialmente conocido como pantalla de nuestra aplicación que hemos creado, es decir la clase java MainActivity, para ello se abre dicho fichero en app>java>org.example.junior.nuevoproyecto, teniendo el siguiente aspecto:

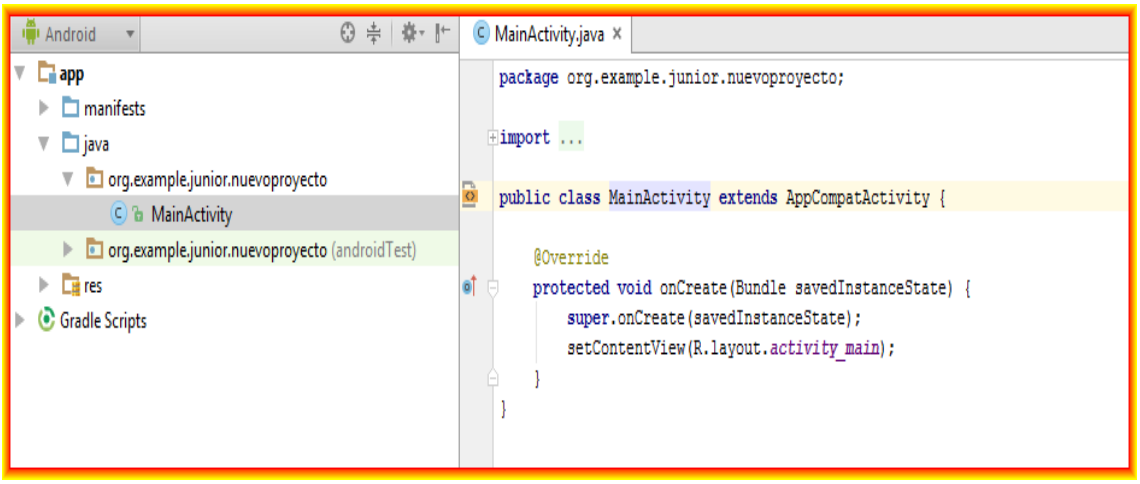


Figura 6-26 Nuestra actividad principal MainActivity.

Podemos ver como la clase MainActivity, extiende (o como ya sabemos, hereda), de AppCompatActivity, y a su vez, dicha clase es descendiente de la clase Activity, se ha de reiterar que una actividad en Android va a estar definida por medio de una clase Java, y va a representar a la pantalla de nuestra aplicación.

Luego de la declaración de nuestra actividad MainActivity, podemos notar la aparición de un término muy especial (concretamente es una anotación) en Android, **@Override**, el cual nos indica de antemano que el método siguiente **onCreate,** va a ser sobrescrito (tal como ya vimos en el capítulo de Java), dicho método va ser el primer en ser invocado por el sistema Android, cuando se ejecute nuestra aplicación o mejor dicho cuando se crea nuestra actividad, este método va a recibir un parámetro de tipo Bundle, éste objeto es útil cuando queremos recuperar la información (por ejemplo cuando rellenamos un formulario) o mejor dicho el estado que teníamos cuando por alguna causa o motivo nuestra aplicación cambia de orientación (horizontal o vertical) o cuando

nuestra aplicación pasa a segundo plano, es decir cuando nuestra aplicación pierde el foco dentro de nuestro móvil. Algo muy importante cuando sobrescribimos un método (en nuestro caso el método onCreate pertenece a la clase padre Activity), es llamarlo desde la clase de la cual hemos heredado dicho método, para referirnos a nuestra clase padre, usaremos la palabra reservada **super**. Finalmente el método onCreate, indica la vista o layout (asociado a la actividad) que se visualizará.

6.4 FICHEROS Y DIRECTORIOS DE UN PROYECTO ANDROID

Como hemos visto en el explorador del proyecto, existe una colección de ficheros y directorios de suma importancia para el funcionamiento de nuestro proyecto o futura aplicación Android. Se podría resumir que un proyecto Android va a estar formado básicamente por un fichero llamado AndroidManifest.xml (que será el descriptor de nuestra aplicación), el código fuente de la aplicación, ficheros con recursos (generalmente imágenes y ficheros .xml) y un fichero llamado Gradle que es mejor ni tocarlo.

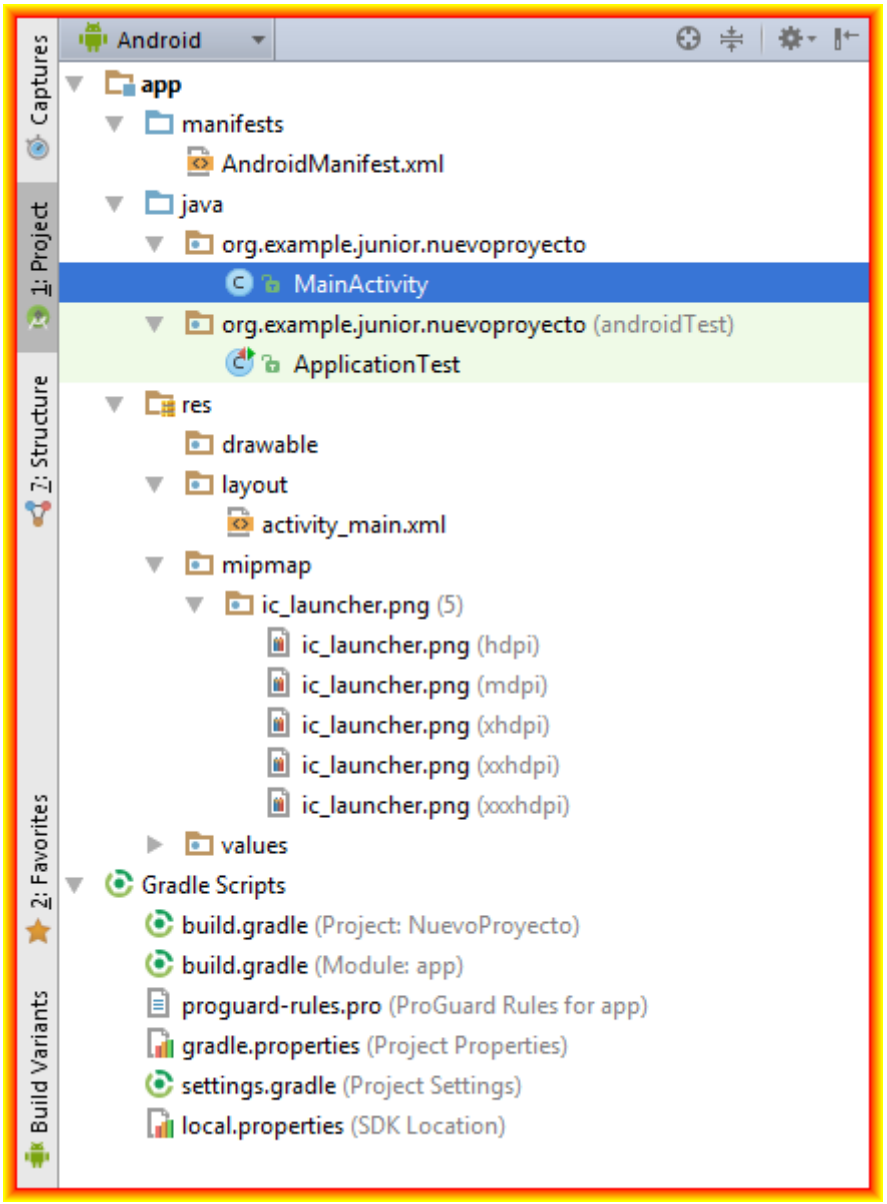


*Figura 6-27 Ficheros y directorios del un proyecto Android (**explorador del proyecto**).*

AndroidManifest.xml: Este fichero con extensión .xml está contenido en una carpeta llamada **manifests**, como ya se comentó, dicho fichero va a describir la aplicación que estemos desarrollando, en él encontraremos mucha información de la aplicación como su nombre e ícono, los intents, actividades (pantallas), temas (colección de propiedades que definen el formato de vistas aplicado a toda la aplicación, muy similares a las hojas de estilos en cascada, CSS, que se aplican en HTML), además desde este fichero se puede activar la opción RTL (Right To Left) para algunos países (en especial países árabes) en el que programar como nosotros lo hacemos, se les hace complicado, ya que en dichas lenguas árabes se lee de derecha a izquierda, y gracias a ésta opción, pueden escribir

código en java de derecha a izquierda. Además de lo comentado anteriormente, gracias a este fichero se declaran los múltiples permisos para poder usar ciertas tecnologias que pueda requerir nuestra aplicación (en el caso de la aplicación de este proyecto final de carrera, hacemos uso de la tecnología Bluetooth, y por ende estamos obligados a declarar dichos permisos de conectividad Bluetooth en el AndroidManifest.xml).

java: en esta carpeta, se va a concentrar todo el código java que se ha utilizado para la creación de nuestra aplicación, dicho código estará contenido en el paquete que hayamos definido durante la creación del proyecto, en nuestro caso el paquete se llama **org.example.junior.nuevoproyecto** (ya se explicó el formato del nombre de paquete en su momento), en dicho paquete estará la actividad principal que se crea automáticamente, así como también las demás clases creadas por nosotros los programadores.

res: En esta carpeta se almacenan los recursos utilizados por la aplicación generalmente imágenes y ficheros XML.

Gradle: En esta carpeta se encuentran los ficheros necesarios para la construcción y compilación del proyecto, personalmente aconsejo nunca tocar ningún elemento de ésta carpeta.

6.5 COMUNICACIÓN BLUETOOTH ENTRE ANDROID Y ARDUINO

Existen muchas alternativas tecnológicas para que los dispositivos móviles puedan recibir y enviar datos, como Wifi, red telefónica, Infrarrojos, Bluetooth, etc. En este proyecto final de carrera se utiliza la tecnología Bluetooth para la transmisión de datos entre el Arduino Mega (o Arduino UNO) y el dispositivo móvil Android, gracias a esta tecnología es posible una comunicación inalámbrica entre distintos dispositivos electrónicos de consumo (como ordenadores, teléfonos móviles, cámaras digitales, impresoras, auriculares y otros dispositivos de reproducción de audio y video, etc.) a una relativa corta distancia (la mayoría de los dispositivos con conectividad Bluetooth, tiene una distancia de hasta 10 metros, aunque si bien es cierto, dependiendo de la potencia utilizada se podría llegar a una distancia máxima de 100 metros, utilizando lo que se conoce como Bluetooth Estándar de clase 1). Bluetooth es una tecnología mundialmente aplicada y universal, con lo que se podrán conectar toda clase de

dispositivos electrónicos sin que haya riesgo de incompatibilidad, aunque también hay que comentar que con Bluetooth la velocidad de transferencia de datos es baja (de 1 a 3 Mbps), a pesar de ello es una tecnología muy sencilla (o mejor dicho fácil de usar), ya que a diferencia de lo que podría ser una Red de Área Local (LAN), Bluetooth no necesita ninguna configuración para establecer una conexión para transferir datos, más características a destacar de esta tecnología serian: la seguridad de la que posee nuestra conexión Bluetooth, esto se consigue ingresando un número de identificación, código PIN, para dicha conexión, evitando así la "presencia" de dispositivos ajenos a nuestra comunicación Bluetooth, también se podría destacar el total control que tenemos sobre dicha conexión, ya que tenemos la opción de aceptar o rechazar la conexión y la transferencia de archivos (a menos que nuestro dispositivo ya esté emparejado con el otro dispositivo), otra característica que se puede resaltar es que el uso de dicha tecnología no conlleva ningún gasto económico, lo único que se necesita es que los dispositivos involucrados tengan soporte Bluetooth para poder comunicarse. Es mejor aclarar que si bien la tecnología Bluetooth proporciona una forma bastante simple para transferir datos, entre dispositivos que gocen de dicha tecnología, no lo es tanto la programación de aplicaciones que se comunican a través de Bluetooth ya que puede llegar a ser muy compleja.

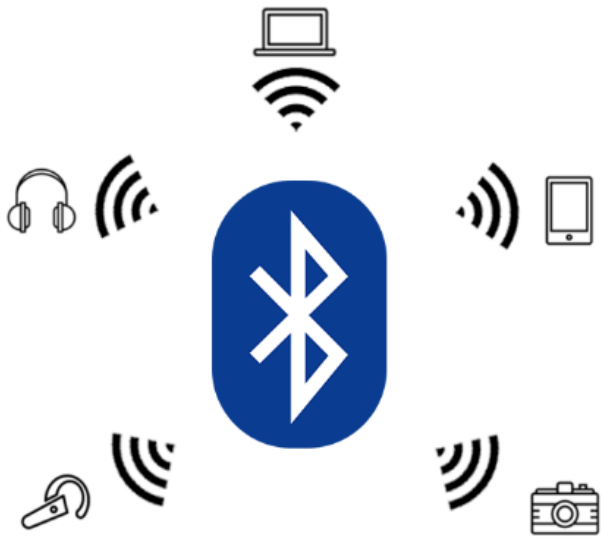


Figura 6-28 Dispositivos Bluetooth.

6.5.1 PASOS A SEGUIR PARA LA COMUNICACIÓN BLUETOOTH

Antes de nada, es conveniente dejar claro que hay dos pasos para lograr una comunicación por medio de la tecnología Bluetooth entre dos dispositivos electrónicos, y éstos pasos son: emparejar (los dos dispositivos "se conocen") y establecer la conexión, hecho estos dos pasos (éstos se detallarán a continuación), recién podemos decir que podemos transmitir datos a través de Bluetooth, en el caso de este proyecto, entre el Arduino Mega (o Arduino UNO) y el dispositivo Android.

6.5.1.1 EMPAREJAR LOS DISPOSITIVOS

Este paso es el más sencillo de los dos y es el que a nivel de usuario todo el mundo conoce, no obstante se detallará.

Antes de que podamos transmitir datos a través de Bluetooth entre dos dispositivos, éstos deben de conocerse el uno al otro, o mejor dicho deben emparejarse, dicho proceso de emparejamiento se puede controlar mediante la interfaz de usuario (lo más sencillo) o mediante la API de Bluetooth Android Java (llegados a este punto, ya sabemos qué es una API).

Si se opta por la primera opción, desde la interfaz de usuario de un teléfono móvil Android o Tablet, nos vamos a Ajustes/Bluetooth (al menos en la mayoría de los dispositivos Android), en un ordenador portátil por ejemplo bastará con pulsar la tecla correspondiente al Bluetooth del teclado y posteriormente haciendo clic en el ícono que debe aparecer a continuación.

Cabe destacar que si lo que queremos es emparejar los dispositivos desde nuestra aplicación, tenemos que hacer uso de la API de Bluetooth Android Java, concretamente tenemos que utilizar el paquete **android.bluetooth** (ya sabemos que son API y paquete).

A continuación se va a describir los pasos que supone emparejar dos dispositivos, para ilustrar estos pasos usaremos un Samsung Galaxy S7 (dispositivo A) y un móvil Huawei P8 lite (dispositivo B):

Debemos encender los adaptadores Bluetooth de ambos dispositivos, al menos en un dispositivo (elegiremos al dispositivo B) debemos activar la opción de

"detectabilidad" o lo que es lo mismo la visibilidad, a fin de que el otro dispositivo (dispositivo A) pueda verlo o detectarlo.

DISPOSITIVO A DISPOSITIVO B

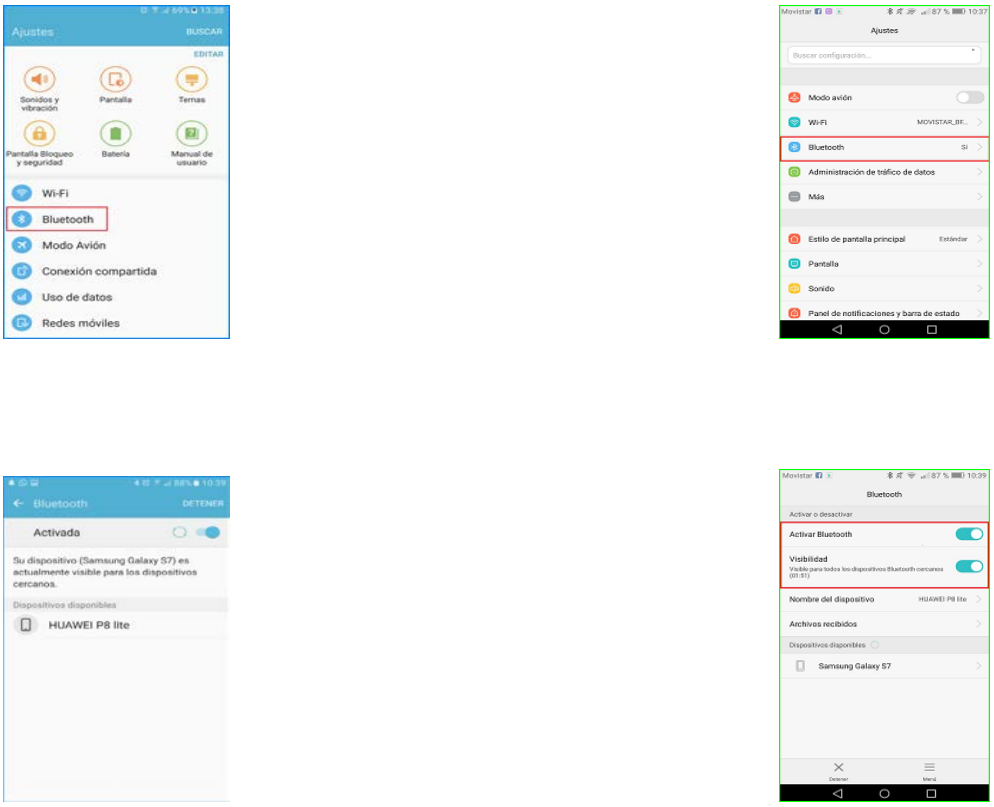


Figura 6-29 Dispositivos activando Bluetooth y habilitando la detectabilidad en uno de ellos.

El dispositivo A manda una solicitud de emparejamiento o vinculación al dispositivo B, dicha solicitud va acompañada de un código de vinculación, que por lo general va a ser generada por el propio sistema, además dicha solicitud también aparece en el propio dispositivo A.

DISPOSITIVO A DISPOSITIVO B

Figura 6-30 Solicitud de vinculación o emparejamiento del dispositivo A al dispositivo B.

Dichos dispositivos quedarán emparejados o vinculados, después de aceptar ambos las solicitudes de emparejamiento o vinculación (se ha aumentado relativamente el tamaño de las imágenes).

DISPOSITIVO A DISPOSITIVO B

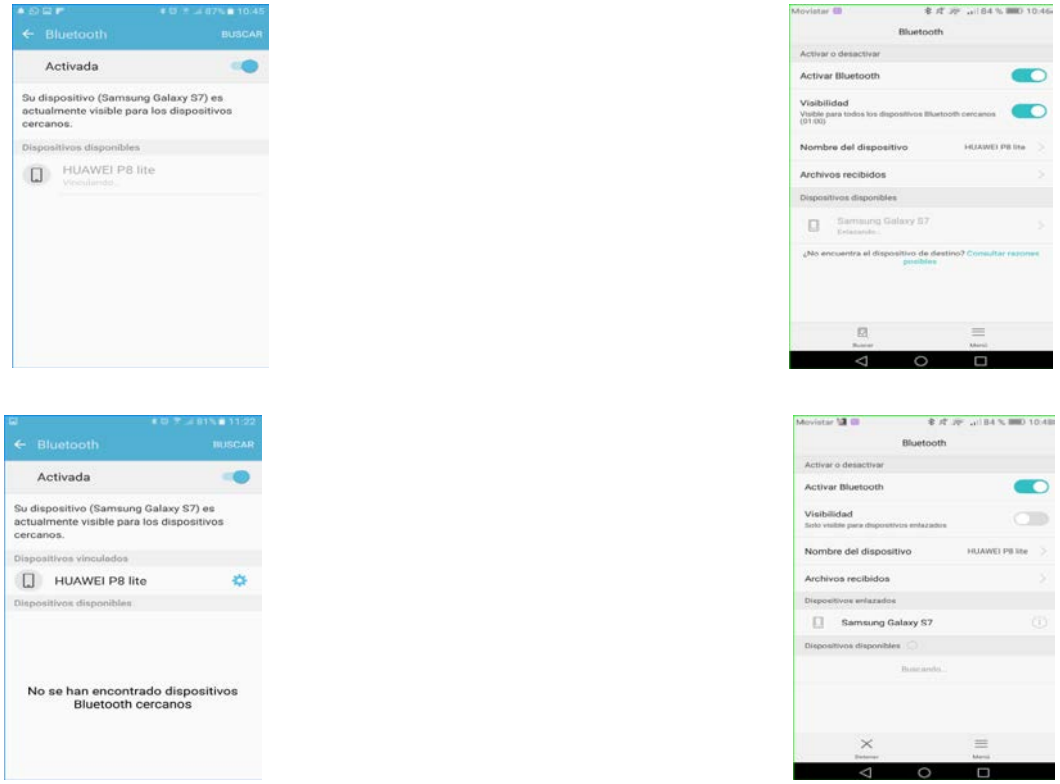


Figura 6-31 Dispositivos A y B vinculados o emparejados.

6.5.1.2 ESTABLECER LA CONEXIÓN ENTRE LOS DISPOSITIVOS

Una vez realizado el paso anterior, es decir ya hemos encendido el adaptador bluetooth, de nuestros dispositivos, y somos capaces de detectar (ver) el otro dispositivo al cual se quiere vincular o emparejar, el siguiente paso es establecer una conexión, para poder transmitir datos entre los dos dispositivos, por medio de un protocolo que usa Bluetooth, y dicho protocolo está basado en un modelo cliente-servidor (el servidor ofrece servicios y el cliente los utiliza) con **sockets** (se explicará enseguida que son los sockets), cabe mencionar que dicho protocolo es muy parecido al protocolo TCP, que se utiliza para programar la transmisión de datos a través de Internet. Vamos a entender por socket, una interfaz a una red de datos (en nuestro caso nuestra red de datos es Bluetooth), eso quiere decir que nuestro programa o aplicación que estemos desarrollando va a leer datos entrantes de un socket (los datos le llegan al socket), y escribir los datos salientes en el mismo socket, desde el punto de vista de desarrollador de aplicaciones Android sólo nos va a importar esto (los sockets), no nos va a importar para nada saber el funcionamiento interno de la red, en nuestro caso Bluetooth, a través del cual se transportan los datos (datos que se transmiten siempre a través de un par de sockets). Como se mencionó, tenemos dos personajes en dicho protocolo, que son el nodo cliente y el nodo servidor, los datos escritos en el socket del nodo cliente son leídos desde el socket del nodo servidor. Podemos ver al socket bluetooth como los extremos de un cable virtual, por el cual se van a transmitir datos (obviamente esto es algo figurativo). En este proyecto, es la placa arduino la que le provee de información, respecto a la temperatura medida, a la aplicación Android, con lo que el arduino será el servidor y la aplicación Android será el cliente, algo que se quisiera destacar respecto a mi proyecto, es que la comunicación no es bilateral, es decir la comunicación va solamente del arduino al dispositivo Android, y con eso es más que suficiente, ya que la finalidad de dicha aplicación Android es la de recibir información del Arduino. En las siguientes líneas se describirán los pasos a seguir, para establecer una conexión entre dos dispositivos Android, que es la forma más general, por que partiendo de esa idea, se logró adaptarla para conseguir una conexión entre el Arduino y la aplicación Android.

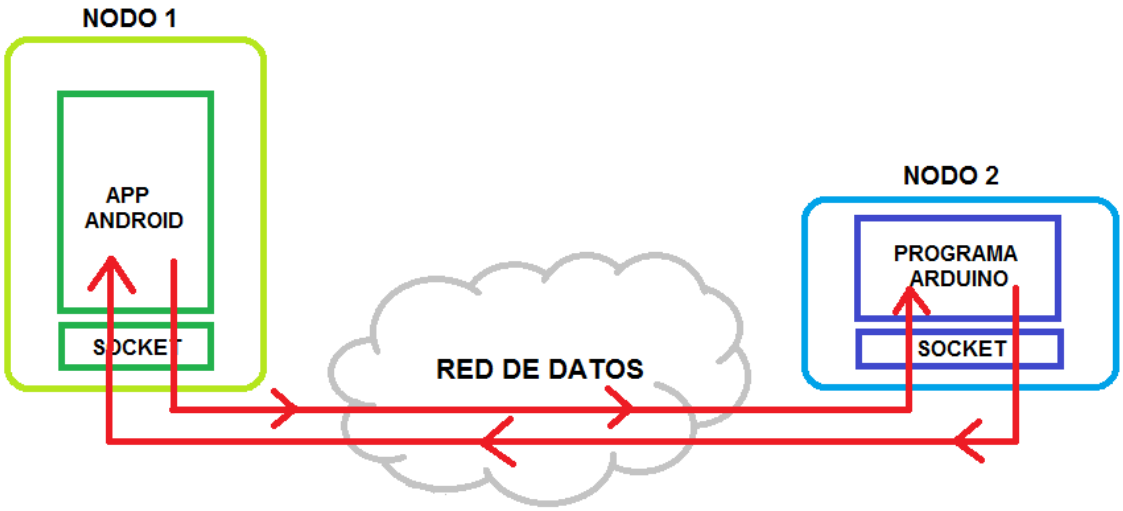


Figura 6-32 Esquema de comunicación basado en sockets entre dos dispositivos.

Viendo este esquema (superior), parece sencillo el tema de los sockets, pero lo cierto es que la programación Android de los sockets es muy compleja, entre otras cosas, conlleva al uso de paquetes Android-Java y por ende de clases definidas para la comunicación Bluetooth. A continuación, vamos a describir los pasos que son necesarios para la transmisión de datos utilizando sockets Bluetooth en Android (en un próximo punto llevaremos esos pasos a la programación en Android):

La aplicación del servidor lo primero que tiene que hacer es crear un socket, llamado **socket servidor** que será identificado mediante un UUID (identificador único universal, que será detallado más adelante), y luego de haber creado dicho socket, el servidor estará listo para esperar solicitudes de conexión por parte de una aplicación cliente.

En el otro lado, se encuentra la aplicación cliente que debe crear un socket, llamado **socket de comunicación del cliente,** y es a través de éste, que envía una solicitud de conexión al servidor.

De nuevo en el lado del servidor, éste debe aceptar dicha solicitud de conexión del cliente, luego de aceptar dicha solicitud, el servidor crea un nuevo socket llamado **socket de comunicación del servidor**. Es a partir de este momento, en el que tanto la aplicación cliente como la aplicación establecen una conexión Bluetooth (comparten un canal de comunicación por radio frecuencia, RFCOMM).

El último paso consiste en que tanto el cliente como el servidor deben obtener los flujos de entrada y salida (E/S) de sus respectivos sockets, debemos pensar que dichos flujos de E/S son como vagones de un tren y los datos son las personas que viajan en ellos. Dicho esto, todos los datos que el cliente escriba en su flujo de salida, aparecerán en el flujo de entrada del servidor y viceversa.

Después de haber descrito los pasos que describen al protocolo Bluetooth para la transferencia de datos, se va realizar un pequeño esquema, a modo de resumen de dichos pasos:

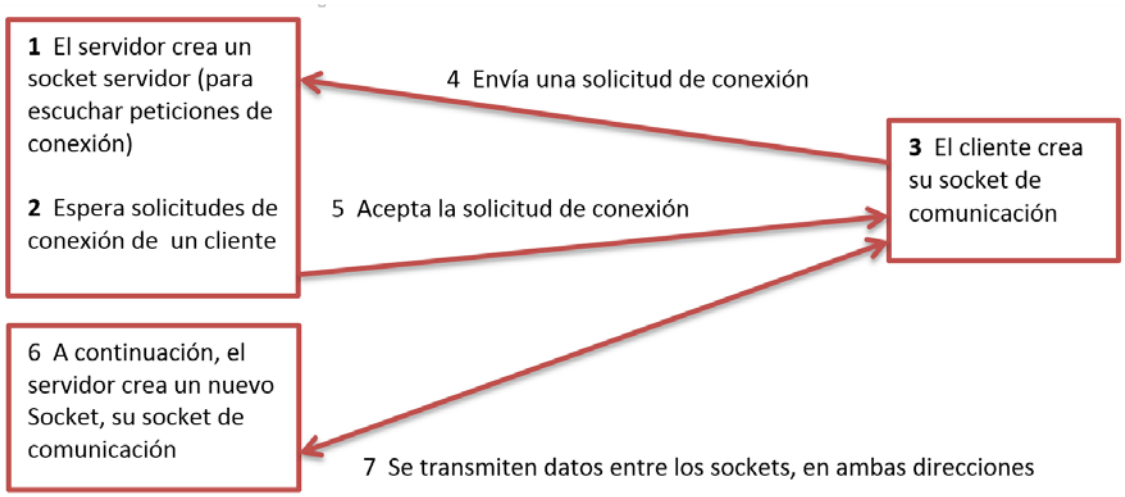


Figura 6-33 Resumen de los pasos necesarios para una comunicación Bluetooth basada en sockets.

Realmente con este esquema, queda muy claro el protocolo general bluetooth, basado en un modelo cliente-servidor, y ya es momento de sacar conclusiones, la primera de todas es que el servidor crea dos sockets, uno para escuchar peticiones de conexión de un cliente y otro para comunicarse con dicho cliente, por su parte, el cliente solo tiene un socket, que le servirá tanto para enviar una solicitud o petición de conexión al servidor como para comunicarse con éste.

La segunda conclusión, es que la comunicación entre los dos sockets del cliente y servidor es bidireccional, es decir en ambos sentidos, con lo que el socket de comunicación del servidor le va a servir tanto para escribir como para leer datos, y lo mismo para el socket de comunicación del cliente.

La tercera conclusión, es que este esquema, es general, con lo que se podría hacer modificaciones, o mejor dicho ser más específico, para usos más concretos, como por ejemplo, el uso que se le he dado para este proyecto final de carrera, en el que solo nos interesa que el arduino "arroje" los valores de las temperaturas obtenidas a la aplicación Android, pensando en estos requerimientos, derivamos en que lo que necesitamos,es sólo una comunicación unidireccional, que vaya del Arduino a la aplicación Android, el que va a ofrecer servicios (dar los valores de la temperatura), es decir, el servidor, va a ser el Arduino, con lo que en la aplicación Android tenemos que implementar toda la programación relacionada con el lado del cliente.

La cuarta conclusión es que, gracias a la experiencia que tiene el autor de estas líneas, programando en Java, sabe que los típicos métodos como connect(), que se usa cuando el cliente manda una petición de conexión al servidor, y el método accept(), que se llama cuando se acepta la conexión, van a bloquear la interfaz de usuario de nuestra aplicación, en resumen van a "congelar" nuestra aplicación, ¿y por qué pasa esto? Simple, primero porque en éste tipo de conexiones nunca existen las respuestas inmediatas, esto quiere decir, por ejemplo que el tiempo que espera el servidor una solicitud de conexión de un cliente, hasta que lo acepta no es inmediato, y además que en el sistema operativo Android, solo existe un hilo (de forma coloquial diremos que es la unidad más pequeña de tarea, el tema de hilos,se abarcará más adelante) , llamado Hilo Principal, que se encarga de atender todos los eventos de interacción del usuario con la pantalla de la aplicación y todo lo que esté pasando dentro del sistema, con lo cual sufre una sobrecarga, generándose un congelamiento de pantalla.

Para solucionar dicho problema, tenemos que pensar en hilos de fondo, o hilos de segundo plano, en su sincronización, es decir que se comuniquen perfectamente entre ellos, lo cual es muy complejo, pero por suerte existe una clase llamada AsyncTask, que va a hacer que el trabajar con hilos sea más sencillo. En resumen la cuarta conclusión es que se tiene que hacer uso de la clase AsyncTask (tarea asíncrona).

La última y quinta conclusión, es que como ya se explicó en su determinado momento, para poder utilizar ciertos servicios como por ejemplo Bluetooth, las aplicaciones Android requieren de permisos específicos, en concreto la aplicación que

se ha desarrollado requiere de éstos dos permisos: **android.permission.BLUETOOTH**, para solicitar y aceptar conexiones, y para lo más importante, que es la transferencia de datos y **android.permission.BLUETOOTH_ADMIN**, para todo lo relacionado a la detección de dispositivos Bluetooth.

6.5.2 EL PAQUETE ANDROID.BLUETOOTH

Tenemos que ver como el paso de establecimiento de conexión entre dos dispositivos, que se explicó anteriormente, se lleva a cabo en Android, y para ello tenemos en la API de Android, un paquete llamado **android.bluetooth** y dicho paquete contiene toda la funcionalidad que necesitamos para utilizar Bluetooth, en dicho paquete hay una serie de métodos con diversas utilidades, que se describen a continuación:

- Activar el Bluetooth en nuestro dispositivo.
- Hacer a un dispositivo detectable (activa la visibilidad bluetooth).
- Buscar o escanear otros dispositivos Bluetooth disponibles.
- Mostrar una lista de los dispositivos bluetooth con los cuales hemos sido emparejados.
- Establecer canales RFCOMM.
- Enviar y recibir datos entre los dispositivos.

Todos estos métodos y sus funcionalidades que se acaba de comentar son contenidos en este paquete, android.bluetooth, por medio de clases diferentes que se va a detallar en breve (recordemos que un paquete contiene clases y que una clase contiene métodos):

La clase **BluetoothAdapter** va a representar al hardware del adaptador bluetooth del dispositivo, dicha clase se utiliza para operaciones de Bluetooth fundamentales. La primera operación es obtener una referencia al adaptador local Bluetooth, y para ello disponemos de un método llamado **getDefaultAdapter()**, la segunda operación es detectar o buscar dispositivos Bluetooth visibles en el entorno), por medio del método **startDiscovery()**, la tercera operación es obtener una lista de los dispositivos con los

dispositivos que ya conocemos, es decir con los que nuestro dispositivo haya sido emparejado, ésta lista la obtenemos gracias al método getBondedDevices(), y la última operación de esta clase consiste en crear el socket del servidor, recordemos que éste se encarga de esperar una solicitud de conexión de un cliente, y esto es gracias al método **listenUsingRfcommWithServiceRecord(String name, UUID uuid)**, este nombre tan largo significa que se crea un socket del servidor que utiliza un canal de comunicación por radiofrecuencia, RFCOMM, y este socket va a estar identificado por un ServiceRecord (registro de servicio), y que queremos escuchar en este socket si tenemos peticiones de conexión de algún cliente, también podemos apreciar que este método tiene dos parámetros, el primero es el nombre del servicio, que podremos elegir libremente (cualquier nombre), el segundo parámetro es el UUID (identificador único universal), que es un conjunto de dígitos hexadecimales, que siguen un patrón 8-4-4-4-12, como se muestra a continuación:

XXXXXXXX-XXXX-**4**XXX-XXXX-XXXXXXXXXXXX

Como se comentó es un patrón 8-4-4-4-12, que consta de cinco grupos de dígitos hexadecimales con 8, 4, 4, 4 y 12 dígitos, respectivamente. Comenzando por la izquierda, el primer dígito del tercer grupo del patrón, define la versión del UUID, sinceramente, creemos que ésta parte es la más confusa de todo el UUID, puesto que se nos dice en la documentación oficial de Android, que para que **dos dispositivos Android se puedan comunicar utilizando los servicios Bluetooth** se debe utilizar la versión **4**, o lo que es lo mismo, poner dicho dígito a **4**, además dicho dígito también significa que el UUID, ha sido elegido aleatoriamente o libremente por el programador, pero esto no es todo, ya que lo que se quiere es una **comunicación entre Arduino y Android**, con lo cual no nos sirve lo del dígito 4, entonces se pudo comprender el porqué la aplicación Android no funcionaba en un primer momento, investigando más en la documentación oficial de Android, la cual se debe de reconocer que es muy amplia y a la vez algo confusa, se nos dice que en el caso de que queramos desarrollar una aplicación Android de tipo cliente (nuestro caso, ya que lo que queremos es que nuestra aplicación se conecte a los servicios de nuestro Arduino Mega) que se conecte a un puerto Bluetooth, y con esto nos da a entender que la comunicación va a ser entre un dispositivo Android y un dispositivo que disponga de un adaptador o modulo Bluetooth con comunicación serial

(nuestro caso) se debe utilizar el siguiente UUID, en la que ya no gozamos de libertad de elección: 00001101-0000-1000-8000-00805F9B34FB, y siempre va a ser la misma para la comunicación entre Android y cualquier dispositivo que disponga de puerto Bluetooth que no sea Android.

La clase **BluetoothDevice**, que como se puede intuir por su nombre, representa al dispositivo Bluetooth en cuestión, es decir podemos referirnos a un dispositivo con esta clase, dicha clase va a ser de mucha utilidad a la hora de crear el socket de comunicación del cliente (recordemos que el socket de comunicación del cliente, envía solicitudes de conexión al servidor y también sirve para comunicarse con éste) gracias a su método **createRfcommSocketToServiceRecord(UUID uuid)**, el nombre de dicho método quiere decir que se crea un socket de cliente que está listo para ser conectado con el socket del servidor con UUID uuid (en nuestro caso el uuid es 00001101-0000-1000-8000-00805F9B34FB). y siempre va a ser la misma para la comunicación entre Android y cualquier dispositivo que disponga de puerto Bluetooth que no sea Android.

La clase **BluetoothDevice**, que como se puede intuir por su nombre, representa al dispositivo Bluetooth en cuestión, es decir podemos referirnos a un dispositivo con esta clase, dicha clase va a ser de mucha utilidad a la hora de crear el socket de comunicación del cliente (recordemos que el socket de comunicación del cliente, envía solicitudes de conexión al servidor y también sirve para comunicarse con éste) gracias a su método **createRfcommSocketToServiceRecord(UUID uuid)**, el nombre de dicho método quiere decir que se crea un socket de cliente que está listo para ser conectado con el socket del servidor con UUID uuid (en nuestro caso el uuid es 00001101-0000-1000-8000-00805F9B34FB).

La clase **BluetoothServerSocket**, representa al socket del servidor, que se encarga de escuchar solicitudes de conexión de algún cliente, y gracias al método de esta clase llamado llamado **accept()**, se acepta peticiones de conexión e inmediatamente después de aceptar petición de conexión, el método **accept()** devolverá un objeto de tipo **BluetoothSocket**.

La clase **BluetoothSocket**, va a representar a los sockets de comunicación, es decir define la interfaz de comunicación a otro dispositivo, ésta clase tendrá dos métodos que

son getInputStream() y getOutputStream(), con los cuales obtenemos los flujos de entrada y salida de información respectivamente para la transmisión de datos (recordar que los flujos son como los vagones del tren y los datos son como las personas que viajan en ellos).

6.6 HILOS EN ANDROID Y LA MEJOR FORMA DE CON ELLOS

Las aplicaciones móviles se convierten con el transcurso del tiempo, en parte de nuestro día a día, y es por ello que los usuarios exigen las mejores prestaciones, no obstante muchas veces, hemos experimentado como algunas de nuestras aplicaciones dejan de funcionar (o al menos eso erróneamente creemos), lo cierto es que no dejan de funcionar, sino que la tarea o tareas que se están llevando internamente en nuestra aplicación requieren de bastante tiempo de ejecución, con lo que nuestra aplicación queda "congelada", impidiendo así que el usuario pueda interactuar con la interfaz gráfica de la aplicación (se pulsa los botones de la aplicación y la pantalla no responde, por ejemplo). Todo ello ocurre porque el encargado de llevar a cabo todas esas tareas o eventos, tiene exceso de trabajo, y dicho encargado es una porción de código, que va a estar asociada a nuestra aplicación, representa la unidad de ejecución más básica, y es conocido como **Hilo Principal** o de **Interfaz de Usuario**, que es el que viene por defecto en el sistema operativo Android (mejor dicho, es el propio sistema Android el que crea el Hilo Principal), y dicho hilo es el encargado de gestionar y responder a dichos eventos, recordemos que en todos los dispositivos basados en Android, la interfaz gráfica de usuario (IGU) cumple un rol importante, ya que siempre interactuamos con ella, escribimos textos, hacemos clics en botones, introducimos patrones de seguridad en la pantalla, activamos el bluetooth, wifi, el modo avión u otro ajuste, etc. Entonces la interfaz de usuario, juega un papel importante en todo dispositivo Android, puesto que en ella se producen los eventos mencionados anteriormente, y las aplicaciones y el propio sistema operativo Android, tienen que responder a dichos eventos.

El sistema operativo Android está basado en eventos, esto quiere decir que responderá a eventos que podrán llegar, en cualquier momento y en cualquier orden.

Los eventos conforme llegan se almacenan en una cola (una cola es una estructura de datos FIFO, First In First Out, que quiere decir: "el primero que llega, es el primero

que sale") y el Hilo Principal será el encargado de atender dicha cola, y ejecutar las tareas correspondientes a cada evento.

Cuando el tiempo de ejecución de las tareas es corto, no es crítico que el hilo de la interfaz de usuario se encargue él solo de todo el trabajo. Sin embargo si la ejecución de dichas necesitara más tiempo, tendríamos problemas (nuestra aplicación se congela), no obstante la solución pasaría por recurrir a los llamados **Hilos de Fondo**, o también conocidos **Hilos en Segundo Plano** o **Hilos Secundarios** (son hilos creados por el programador), pero esto se explicará en breve así como la utilización de una clase muy útil llamada AsyncTask que nos permitirá controlar y hacer más fácil la comunicación entre los distintos hilos que utilicemos.

6.6.1 HILOS EN ANDROID

Como bien lo explicaba anteriormente, cuando una aplicación "arranca" o se inicia, va a crear un hilo de ejecución, llamado hilo principal o de interfaz de usuario, el cual es muy importante, ya que es el encargado de gestionar y responder a los eventos, e incluso de dibujar la interfaz e interactuar con el usuario, por lo tanto, no es conveniente bloquearlo con tareas muy costosas (tareas que generan mucha carga al sistema), o como resultado, tendremos un error como el que se muestra en la siguiente imagen:

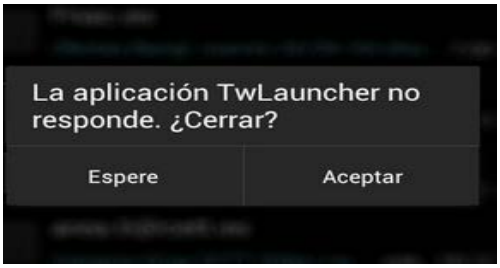


Figura 6-34 Consecuencias de tener el Hilo Principal sobrecargado de tareas.

Cuando nos referimos a una operación muy costosa por ejemplo podría ser una consulta en una base de datos, o una conexión remota con un dispositivo Android, una operación matemática muy costosa como por ejemplo el factorial del número 89637, generándose así lo que se conoce como bloqueo de la aplicación (app) o aplicación "congelada". Dicho problema genera el descontento del usuario y por ende la muy probable desinstalación de la app.

Para solucionar esto, se va a crear hilos independientes llamados Hilos de Fondo o Secundarios que se ejecuten al mismo tiempo que el principal, sin bloquearlo.

Estos hilos tienen dos características muy importantes:

- **No bloquean la interfaz de usuario**.
- **No tienen acceso directo al hilo de la interfaz de usuario.**

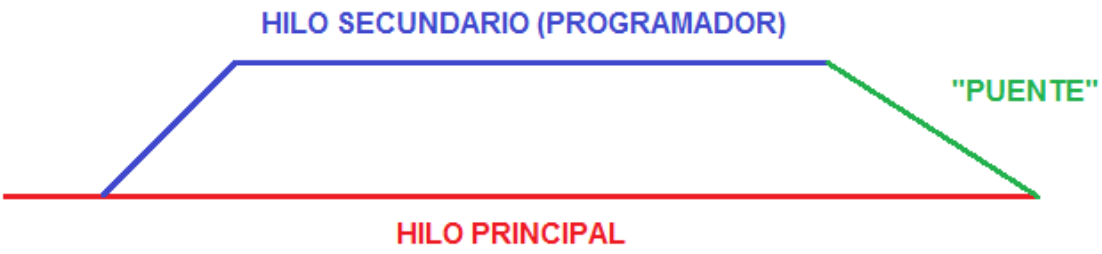


Figura 6-35 Hilo Principal y Secundario trabajando en paralelo (multithread).

En definitiva, tenemos dos tipos de hilos, el Hilo (Hilo Principal), que ya viene por defecto, el cual es creado por el sistema operativo Android, cada vez que se inicia una nueva aplicación; y el Hilo Secundario o Hilo en Segundo Plano, que va a facilitar con las tareas pesadas a nuestro Hilo Principal, como ya se mencionó este Hilo Secundario, no puede bloquear la interfaz de usuario, básicamente porque no tiene acceso a ella, esto en sí es malo, ya que esto implica que no hay una comunicación directa entre éstos dos hilos (Principal y Secundario), por lo que se sugiere una serie de alternativas, que sirvan de mediador o "puente" entre éstos dos hilos, algunas de dichas alternativas serian: el uso de la **Clase Handler**, y el uso de los métodos **post** (para actuar sobre cada componente de la interfaz de usuario), y **runOnUiThread()** (para que se pueda enviar operaciones al hilo principal desde el hilo secundario), de todas maneras estas alternativas han sido descartadas, puesto que suponen una alta complejidad del código en la aplicación que se ha desarrollado, además y para empeorar las cosas, estos métodos no tienen en cuenta la sincronización entre hilos secundarios (en muchos casos nos va a interesar crear varios hilos secundarios, los cuales tienen que estar comunicándose entre ellos y evitar que accedan a zonas de datos al mismo tiempo, ya que esto supondría un gran problema para la aplicación, ya que los datos se podrían

incluso perder sin control), por suerte existe otra gran alternativa la cual es la clase **AsyncTask**, que nos va a permitir el uso, gestión y comunicación entre hilos de una forma muy sencilla.

6.6.1.1 CLASE ASYNCTASK

Además de los problemas que se comentó sobre las alternativas (descartadas) que sirven de "puente" entre el hilo de la interfaz de usuario y el hilo secundario, está el hecho de que dicho código del programa o aplicación (haciendo uso de esas alternativas) es complicado de leer, y pero aún si nuestra aplicación dispone de varios controles en su interfaz, ya que la actualización de las mismas no sería tan simple y peor aún si nuestra aplicación sufre de una gran interacción con el usuario interacción, el código empezaría a ser difícilmente manejable, complicado de leer y mantener, y por tanto también más propenso a errores. Todo ello parece realmente malo y es que en verdad lo es, pero es aquí donde Android, o mejor dicho Google (Google es dueño de Android) llega en nuestra ayuda y nos brinda la **clase AsyncTask**, que entre otras cosas, nos va a permitir realizar lo mismo que esas otras alternativas (los llamado coloquialmente "puente") pero con la ventaja de no tener que utilizar ciertos "apaños" como los métodos **post o runOnUiThread()** y lo más importante de todo, de una forma mucho más organizada y entendible. La manera correcta y básica de utilizar la clase AsyncTask consiste en crear una clase que herede o extienda de ella y sobrescribir varios de sus métodos entre los que repartiremos la funcionalidad de nuestra tarea. Dicho métodos son los siguientes:

- **onPreExecute()**: Es en este método donde tenemos que realizar los trabajos previos a la ejecución de nuestra tarea (la tarea costosa). Se suele utilizar normalmente para para configurar la tarea, inicializar la interfaz (aunque de esto se puede encargar el método onCreate de la clase principal), este método es muy poco empleado, de hecho en este trabajo no lo utilizamos.
- **doInBackground()**: Este método va a contener el código principal de nuestra tarea, en dicho método se va a definir las operaciones que serán ejecutadas en segundo plano, de tal manera que su código es ejecutado en un hilo secundario de manera concurrente al hilo principal.

- **onProgressUpdate()**: Se ejecutará en respuesta al método publishProgress() desde el método doInBackground() (el método publishProgress ,es llamado por el hilo en segundo plano para indicar su progreso, aunque es opcional).
- **onPostExecute()**: Se ejecutará desde el hilo principal cuando finalice nuestra tarea, o dicho de otra forma, tras la finalización del método doInBackground().
- **onCancelled()**: Este método se ejecutará cuando se cancele la ejecución de la tarea antes de su finalización normal.

A continuación se muestra una estructura de dicha clase, donde se proporciona tres tipos genéricos para:

- Los parámetros recibidos desde el hilo de la interfaz de usuario.
- Los valores informando sobre el progreso de la operación.
- El resultado devuelto al hilo de la interfaz de usuario.

```
class MiTareaAsincrona extends AsyncTask <Parametros,Progreso,Resultado>{

  @Override
  protected void onPreExecute() {
          …
  }
  @Override
  protected Resultado doInBackground(Parametros... par) {
          …
  }
  @Override
  protected void onProgressUpdate(Progreso... prog) {
           …
  }
  @Override
  protected void onPostExecute(Resultado result) {
          …
  }
}
```

Figura 6-36 Estructura de una clase hija de AsyncTask.

CAPÍTULO 7. EXPLICACIÓN DEL PROGRAMA ANDROID

Código en Anexo 1.

El programa o aplicación Android que se ha desarrollado, básicamente consta de cuatro clases, las cuales describiré a continuación:

- **MainActivity:** Es nuestra clase principal, en ella se define nuestra interfaz gráfica, es decir cada vista o elemento visual de nuestra aplicación (en nuestro caso, dos TextView (uno de ellos para mostrar la temperatura y el otro para mostrar el instante de tiempo en el que se registra cada valor de temperatura) y un botón (el botón que muestra una breve información de la aplicación), además es en esta clase donde se va a llevar todo el proceso de activación bluetooth de nuestro dispositivo Android, es decir se comprueba que nuestro dispositivo disponga de conectividad bluetooth (Obtenemos el adaptador Bluetooth de nuestro dispositivo), en segundo lugar se comprueba que Bluetooth esté activado, y en tercer lugar se recorre la lista de los dispositivos emparejados previamente hasta encontrar el dispositivo que nos interesa (en nuestro caso es el dispositivo de nombre MODULO BLUETOOTH JUNIOR). Pero lo realmente importante de esta clase es que es en ésta donde se crea la tarea asíncrona y se **arranca dicha tarea** pasándole como parámetro información del dispositivo con el cual estamos emparejado, por medio del método execute().
- **MiTareaAsync:** Esta clase, es una clase hija de la clase AsyncTask, con lo cual ésta clase nos ayuda a soportar la programación concurrente al hilo de la interfaz de usuario, concretamente esta clase nos ayuda en la ejecución de tareas en segundo plano, pudiendo así devolver el resultado de dicha tarea costosa al hilo de la interfaz de usuario para que éste "pinte" dicho resultado en la interfaz gráfica de usuario. Además es de vital importancia mencionar que es en esta clase donde se lleva a cabo la creación de los sockets de comunicación y todo el protocolo necesario para la transmisión de datos a través de bluetooth.
- **Temperatura:** Esta clase realmente es muy sencilla, no hereda de ninguna otra clase y va a estar formada únicamente por métodos GET y SET (los cuales

fueron explicados en su momento), y nos va a ser de mucha utilidad para poder transmitir información desde la tarea asíncrona a la actividad principal.

- **AcercaDe:** Esta clase de propósito explicativo, al pulsar un botón se nos muestra un mensaje que describe la aplicación a grandes rasgos.

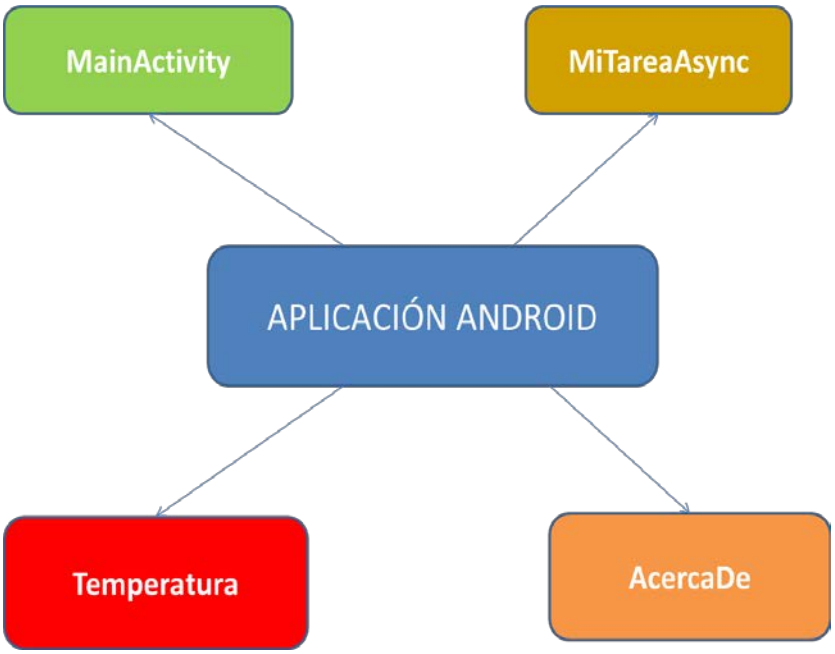


Figura 7-1 Estructura de la aplicación Android.

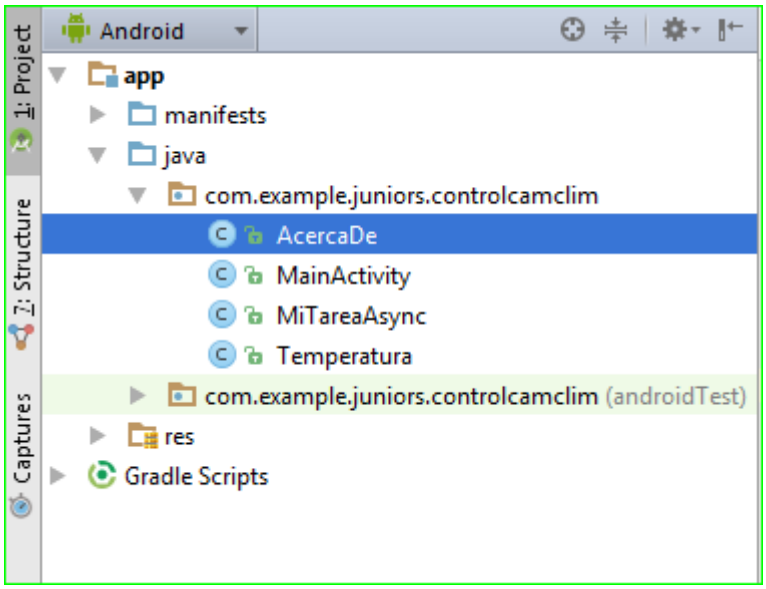


Figura 7-2 Estructura de la aplicación Android, en el explorador de proyecto de Android Studio.

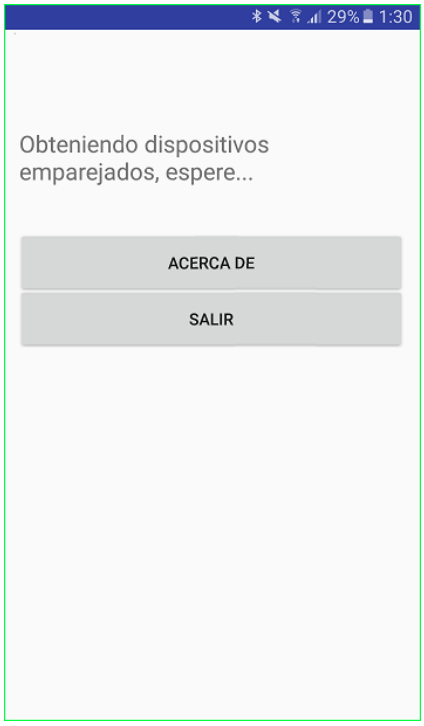
Después de que se han descrito las cuatro clases principal, y cuyos códigos de programación se encuentran en el anexo 1, el resumen seria el siguiente: la clase principal va ser la responsable del aspecto de la aplicación y de muchas acciones (ya han sido comentadas en su descripción), la clase MiTareaAsync se va a encargar del la verdadera y mas importante tarea de toda la aplicación, que es la de recoger los datos (valores de temperaturas) del arduino y mostrarlos en la pantalla de la aplicación, pero esta clase necesita de la ayuda de la clase Temperatura, ya que esta clase, es un parámetro de la clase MiTareaAsync, finalmente la clase AcercaDe, que es la clase de carácter informativo acerca de la aplicación.

CAPÍTULO 8. DEMOSTRACIÓN DE FUNCIONAMIENTO

Ya habiéndose explicado la aplicación Android en el capítulo 7, ahora se pasarán a mostrar unas imágenes en la cual se demustra que efectivamente la aplicación funciona:

Al arrancar la aplicación, se muestra el mensaje de activar bluetooth, el cual si no está activado nos obliga a hacerlo, dando por supuesto que previamente hemos emparejado nuestro modulo blluetooth HC-05 con el dispositivo móvil donde se esté ejecutando nuestra aplicación.

Figura 8-1 Petición de activación Bluetooth para poder utilizar la aplicación.



Después de haberse emparejado los dispositivos, y activado el adaptador bluetooth en nuestro dispositivo móvil, la aplicación gracias a la clase MainActivity obtiene la lista de los dispositivos que han sido emparejados con nuestro dispositivo móvil y selecciona el deseado (HC-05).

Figura 8-2 Obtencion de los dispositivos emparejados.

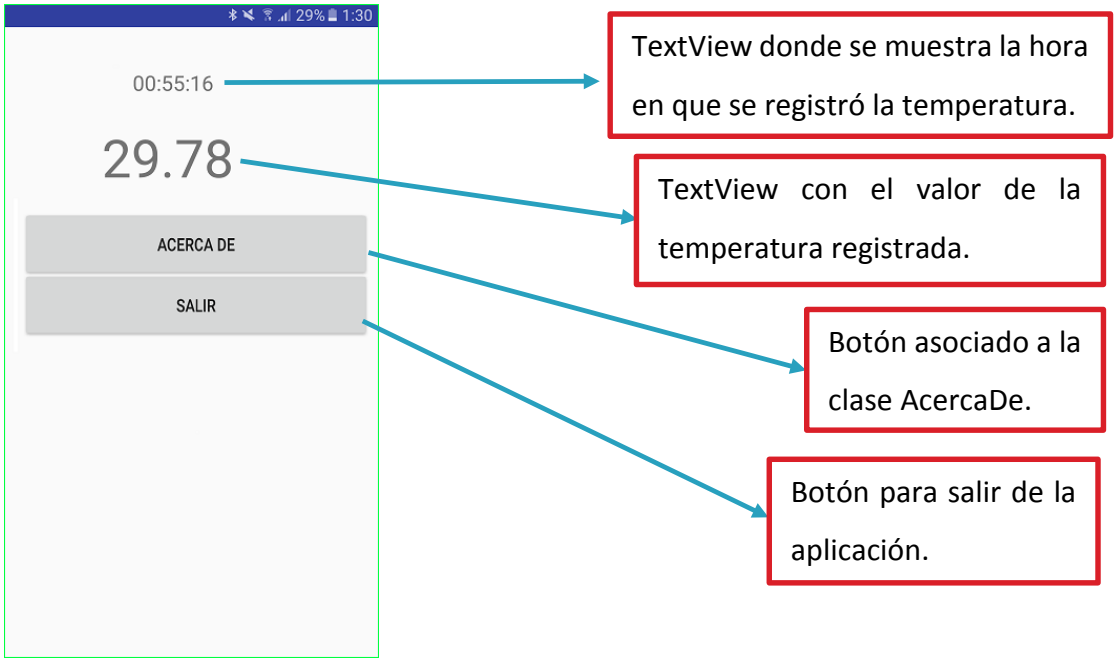


Figura 8-3 Partes de la interfaz de la aplicación.

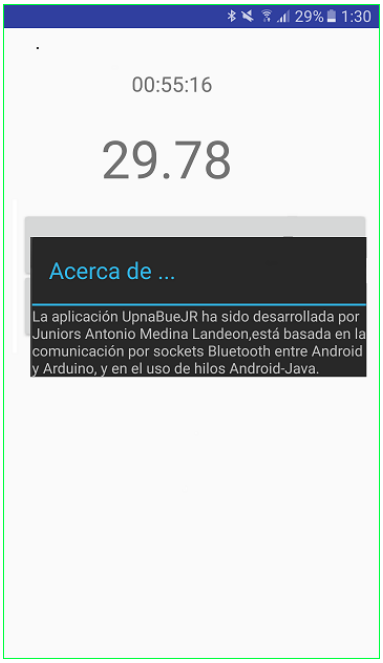


Figura 8-4 Menú informativo que se muestra al pulsar el botón ACERCA DE.

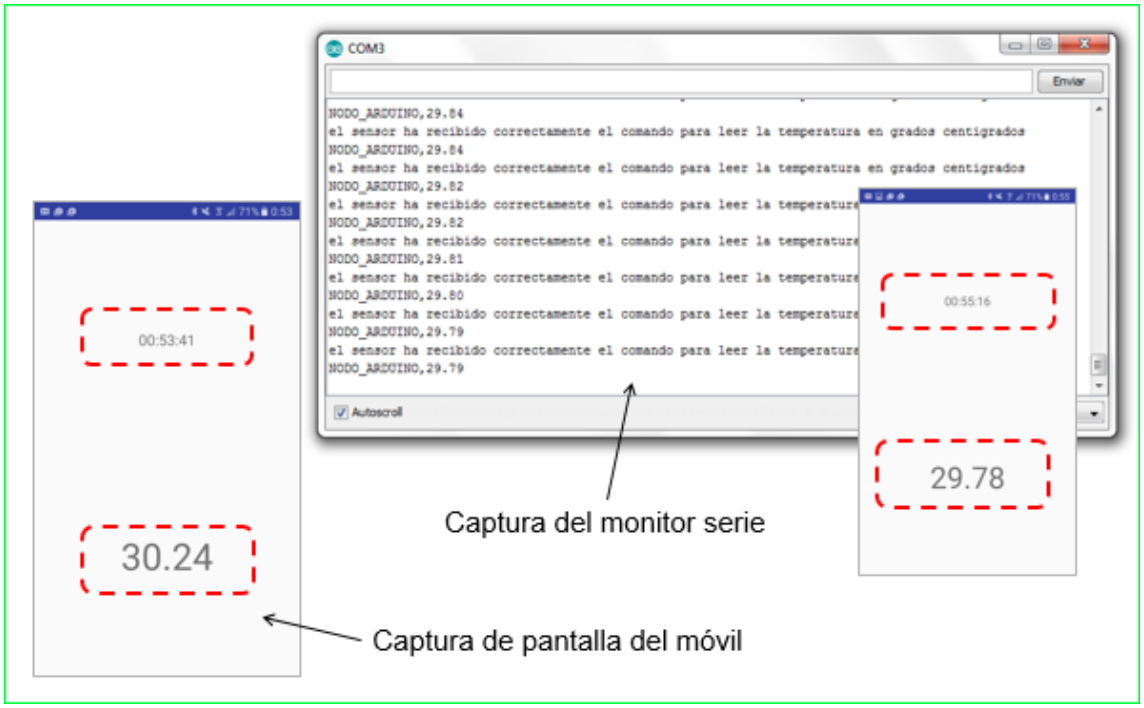


Figura 8-5 Demostración final del proyecto (los valores que aparecen en el monitor serie del Arduino se ven reflejados en la aplicación que hemos desarrollado).

CONCLUSIONES Y LINEAS FUTURAS

A modo de conclusiones de este proyecto se pueden destacar los siguientes puntos:

- Se ha conseguido analizar los componentes tanto físicos como informáticos necesarios para el establecimiento de una comunicación bluetooth entre el microcontrolador encargado del gobierno de la cámara climática y un dispositivo móvil.
- Se ha programado el microcontrolador Arduino para la interrogación del módulo sensor XX… Se trata de una comunicación serie no estándar por lo que ha sido necesario analizar las estructuras de las tramas de datos que son necesarias para la comunicación con el módulo sensor.
- Se ha seleccionado y configurado un módulo de transmisión Bluetooth para la comunicación del Arduino con el dispositivo móvil.
- Se ha desarrollado una aplicación Android que es capaz de conectarse con el microcontrolador arduino y recibir los datos de temperatura que éste envía en tiempo real.
- Para ello se han resuelto problemas diversos. En primer lugar la mayor parte de la bibliografía existente documenta la programación de sistemas de comunicaciones bluetooth entre dos dispositivos Android. Sin embargo no es trivial la adecuación de este sistema de comunicación a una configuración Android-microcontrolador.

A continuación se comentarán algunas potenciales líneas de trabajo que quedan abiertas para futuros proyectos, que por motivos de extensión y tiempo no se han abordado en el presente trabajo.

- Finalización de la cámara climática completando la parte de electrónica de potencia y del sistema de regulación automático de la temperatura.
- Mejorar la interfaz gráfica
- Establecimiento de una comunicación bidireccional. Esto permitiría que desde el dispositivo Android se pudieran enviar comandos al microcontrolador y así poder tener un control global.

BIBLIOGRAFÍA

Fuentes para programación en Android:

EL GRAN LIBRO DE ANDROID Jesús Tomás Girones

http://www.androidcurso.com/

https://developer.android.com

https://developer.android.com/studio/index.html (para tener el Android Studio)

http://iit.qau.edu.pk/books/Android.Application.Development.for.For.Dummies.pdf

Fuentes para Arduino:

LIBRO Arduino-curso práctico de formación Oscar Torrente

https://www.arduino.cc/

http://personales.upv.es/moimacar/master/download/arduino.pdf

Fuente para sensor SHT15:

https://www.sparkfun.com/datasheets/Sensors/SHT1x_datasheet.pdf

Fuentes para módulo Bluetooth HC-05:

http://www.geekfactory.mx/tutoriales/bluetooth-hc-05-y-hc-06-tutorial-de-configuracion/

https://www.youtube.com/watch?v=XXtf1XtQC_0 (el mejor video que pude ver)

Fuentes para programación en Java:

http://www.tutorialspoint.com/java/java_tutorial.pdf

http://www.cloudbus.org/~raj/java/Chapter13.pdf (para el tema de sockets)

www.edu4**java**.com/es/index.html

http://docs.oracle.com/javase/tutorial/java/

Fuentes para temas de curiosidad:

http://www.androidauthority.com/want-develop-android-apps-languages-learn-391008/

http://www.elandroidelibre.com/2014/03/las-mejores-aplicaciones-y-herramientas-para-desarrollo-web-con-android.html

https://www.330ohms.com/blogs/blog/115110980-el-ide-de-arduino-cc-y-arduino-org-son-lo-mismo

http://es.norton.com/android-vs-ios/article

http://www.xataka.com/makers/13-proyectos-asombrosos-con-arduino-para-ponerte-a-prueba-y-pasar-un-gran-rato

Y varias webs sueltas de java y Android además de varios videotutoriales de YouTube.

ANEXO 1

Código de Android

CLASE MainActivity

```
package org.example.junior.upnabluejrs;

import android.app.Activity;
import android.app.AlertDialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.DialogInterface;
import android.content.Intent;
import android.graphics.drawable.AnimationDrawable;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;

import java.util.Set;

public class MainActivity extends Activity implements
MiTareaAsync.Escuchador {
    private static final String TAG = "MainActivity";
    private static final int REQUEST_ENABLE_BT = 1;
    private static final String NOMBRE_DISPOSITIVO_BT = "MODULO
BLUETOOTH JUNIOR";//Nombre de nuestro dispositivo bluetooth
    private TextView temperatura_TextView;
    private TextView hora_temperatura;
    private MiTareaAsync tareaAsincrona;
    private ImageView imagen;
  /* METODO ONCREATE:
    /* éste método es llamado al crearse por primera vez la
actividad*/
    // Se ejecuta en cuanto se crea la actividad, es el lugar ideal
para inicializar la actividad, es decir para crear las vista o grupo
de vistas(layout) que contendrá (interfaz de usuario) y algunas
variables de utilidad.
    @Override
    protected void onCreate(Bundle savedInstanceState) {
    /*Inicializamos la activity e inflamos el layout*/
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        /*Obtenemos las referencias a los dos text views que usaremos
para "pintar" la temperatura*/
        temperatura_TextView = (TextView)
findViewById(R.id.textView_Temperatura);//Mostrará la temperatura
        hora_temperatura =
(TextView)findViewById(R.id.textView_Hora);//Mostrará la hora a la que
fue registrada
        /*imagen=(ImageView)findViewById(R.id.imageView2);
        final AnimationDrawable myAnimationDrawable=
                (AnimationDrawable)imagen.getDrawable();
        imagen.post(
                new Runnable(){
                    @Override
```

```
                    public void run() {
                        myAnimationDrawable.start();
                    }
                }); */

    }
    @Override
    protected void onResume() {
    /* El metodo on resume es el adecuado para inicialzar todos lo
procesos que vayan a actualizar la interfaz de usuario
    Por lo tanto invocamos aqui al método que activa el BT y crea la
tarea asincrona que recupera los datos*/
        super.onResume();
        descubrirDispositivosBT();
    }
    private void descubrirDispositivosBT() {
    /*
    Este método comprueba si nuestro dispositivo dispone de
conectividad bluetooh.
    En caso afirmativo, si estuviera desctivada, intenta activarla.
    En caso negativo presenta un mensaje al usuario y sale de la
aplicación.
     */
        //Comprobamos que el dispositivo tiene adaptador bluetooth, o
mejor dicho obtenemos el adaptador bluetooth
        BluetoothAdapter adaptador_bluetooth =
BluetoothAdapter.getDefaultAdapter();
  //Si el dispositivo no soporta bluetooth --> mensaje al usuario de
3.5 segundos y salimos de la aplicación
        hora_temperatura.setText("Comprobando bluetooth");
        if (adaptador_bluetooth != null) {
            //El dispositivo tiene adapatador BT. Ahora comprobamos
que bt esta activado.

            if (adaptador_bluetooth.isEnabled()) {
                //Esta activado. Obtenemos la lista de dispositivos BT
emparejados con nuestro dispositivo android.
                hora_temperatura.setText("Obteniendo dispositivos
emparejados, espere...");
                Set<BluetoothDevice> pairedDevices =
adaptador_bluetooth.getBondedDevices();
                //Si hay dispositivos emparejados
                if (pairedDevices.size() > 0) {
                    /*
                    Recorremos los dispositivos emparejados hasta
encontrar el adaptador BT del arduino, en este caso se llama MODULO
BLUETOOTH JUNIOR
                     */
                    BluetoothDevice arduino = null;
                    for (BluetoothDevice device : pairedDevices) {
                        if
(device.getName().equalsIgnoreCase(NOMBRE_DISPOSITIVO_BT)) {
                            arduino = device;

                        }
                    }
                    if (arduino != null) {
                        tareaAsincrona = new MiTareaAsync(this);
                        tareaAsincrona.execute(arduino);
                    } else {
                        //No hemos encontrado nuestro dispositivo BT,
```

```
es necesario emparejarlo antes de poder usarlo.
                        //No hay ningun dispositivo emparejado.
Salimos de la app.
                        Toast.makeText(this, "No hay dispositivos
emparejados, por favor, empareje el arduino",
Toast.LENGTH_LONG).show();
                        finish();
                    }
                } else {
                    //No hay ningun dispositivo emparejado. Salimos de
la app.
                    Toast.makeText(this, "Juniors,no hay dispositivos
emparejados, por favor, empareje el arduino",
Toast.LENGTH_LONG).show();
                    finish();
                }

            }else {
                muestraDialogoConfirmacionActivacion();
  // Se nos mostrará un mensaje obligándonos a activar bluetooth, con
lo cual damos en si, y se nos cierra la aplicación, para poder activar
bluetooth en nuestro móvil, posteriormente volvemos a abrir nuestra
aplicación
            }
        }else {
            // El dispositivo no soporta bluetooth. Mensaje al usuario
y salimos de la app
            Toast.makeText(this, " Juniors el dispositivo no soporta
comunicación por Bluetooth", Toast.LENGTH_LONG).show();
        }
    }@Override
         protected void onStop() {
        /*Cuando la actividad es destruida, se ejecuta este método.
        Es el lugar adecuado para terminar todos aquellos procesos que
se ejecutan en segundo plano, como es el caso de nuestra tarea
asíncrona que actualiza la interfaz de usuario.
        */
        super.onStop();
        if (tareaAsincrona != null) {
            tareaAsincrona.cancel(true);
        }
    }
    /*Los métodos onTaskCompleted, onCancelled,
onTemperaturaActualizada, son lo que se conocen como "escuchadores".
   */
    @Override
    public void onTaskCompleted() {
    }
    @Override
    public void onCancelled() {

    }
    @Override
    public void onTemperaturaActualizada(Temperatura p) {
        temperatura_TextView.setText(p.getTemperatura());
        hora_temperatura.setText(p.getInformacion());
    }
    private void muestraDialogoConfirmacionActivacion() {
        new AlertDialog.Builder(this)
                .setIcon(android.R.drawable.ic_dialog_alert)
                .setTitle("Activar Bluetooth")
```

```
                .setMessage("BT esta desactivado. ¿Desea activarlo?")
                .setPositiveButton("Si", new
DialogInterface.OnClickListener(){
                    @Override
                    public void onClick(DialogInterface dialog, int
which) {
                        //Salimos de la app
                        finish();
                     }
                })
                .show();

    }
    //Éste metodo (escuchador de evento) se ejecutará cuando se pulse
el boton ACERCA DE LA APLICACIÓN
    public void lanzarAcercaDe(View view){
        Intent i = new Intent(this, AcercaDe.class);
        startActivity(i);
    }
    public void salir(View view){
        finish();
    }
    }
```

CLASE MiTareaAsync

```
   package org.example.junior.upnabluejrs;

import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.os.AsyncTask;

import java.io.BufferedReader;
import java.io.Closeable;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.UUID;

/**
 * Created by JUNIORS on 06/09/2016.
 */
public class MiTareaAsync extends AsyncTask<BluetoothDevice,
Temperatura, Void>
        {
         private static final String TAG = "MiTareaAsync";
        //Identificador unico universal del puerto bluetooth en
android (UUID)
        private static final String UUID_SERIAL_PORT_PROFILE =
"00001101-0000-1000-8000-00805F9B34FB";
        private Temperatura temperatura = new Temperatura();
          //declaro y creo un objeto de la clase Temperatura
  // en ésta clase asincrona voy a crear los sockets necesarios para
la comunicación y todo lo que concierne a ello
        private BluetoothSocket mSocket = null;
  //necesito recoger el flujo de entrada por medio de un buffer
        private BufferedReader mBufferedReader = null;
  // lo utilizo para controlar la entrada mediante buffer
        private Escuchador callback;
  //creo el objeto formato de tiempo de la clase SimpleDateFormat
        private SimpleDateFormat sdf = new
SimpleDateFormat("HH:mm:ss");
        private boolean recibiendo = false;
        private InputStream flujo_entrada = null; // flujo de entrada
        private InputStreamReader lectura_entrada = null;//para leer
los bytes de un flujo de datos y convertirlos a caracteres UNICODE
        private int contadorConexiones = 0;
        public interface Escuchador {
        void onTaskCompleted();
        void onCancelled();
        void onTemperaturaActualizada(Temperatura p);
        }
        public MiTareaAsync(Escuchador CALLBACK) {
        callback = CALLBACK;
        }
        @Override
        protected Void doInBackground(BluetoothDevice... devices) {

        final BluetoothDevice device = devices[0];
        //Realizamos la conexion al disp.blueetoth.
```

```
          //mientras que la tarea en segundo plano no ha sido
cancelado seguiremos recibiendo informacion del arduino
        while (!isCancelled()) {
        if (!recibiendo) {
        recibiendo = conectayRecibeBT(device);
        }
        }
        cierra();
        return null;
        }
        private boolean conectayRecibeBT(BluetoothDevice device) {
        //Abrimos la conexión con el dispositivo.
        boolean ok = true;
        try {
        contadorConexiones++;
          //creo el socket de comunicación
        mSocket = device.createRfcommSocketToServiceRecord(uuid());
        mSocket.connect();
         //creo los flujos de entrada, el lector y el buffer
        flujo_entrada = mSocket.getInputStream();
        lectura_entrada = new InputStreamReader(flujo_entrada);
        mBufferedReader = new BufferedReader(lectura_entrada);
        temperatura.setInformacion("Sin datos...");
        publishProgress(temperatura);
         //informamos del progreso, en este caso el valor de la
temperatura
       //mientras no se cancela la tarea asincrona (ASYNCTASK),
pregunto por el valor de temperatura

        while (!isCancelled()) {
        try {
        String aString = mBufferedReader.readLine();
        if ((aString != null) && (!aString.isEmpty())) {
          //si se cumple las dos condiciones de arriba vusualizamos el
tiempo del registro de la temperatura
        //tiempo en que se registra un dato.
        temperatura.setInformacion(sdf.format(new Date()));
          //Formatear la fecha usando el formateador y pasándole como
argumento el objeto Date

      //Por parte del arduino voy a recibir la informacion"NOMBRE DEL
DISPOSITIVO BLUETOOTH,VALOR DE LA TEMPERATURA"
                        //OJOOOO notar que hay una coma de por medio
                        //Por lo tanto tendré que usar el método split
        try {
        String s[] = aString.split(",");
  //el split hace un rompimiento por medio de la coma
        temperatura.setNombreDispositivo(s[0]);
        temperatura.setTemperatura(s[1]);
        publishProgress(temperatura);
        } catch (Exception e) {
        //Si falla el formateo de los datos, no hacemos nada.
Mostramos la excepción en la consola para
        //observar el error.
        e.printStackTrace();
        }
        }
        } catch (IOException e) {
        e.printStackTrace();
        }
        }
```

```
        /* Si el Asynctask se cancela , cerramos la conexión bluetooth
*/

        temperatura.setInformacion("Cerrando conexion BT");
        } catch (IOException e) {
        ok = false;
        e.printStackTrace();
        temperatura.setInformacion("Error conectando con dispositivo
bt, reintento " + contadorConexiones + "... Si este error se repite,
reinicie el arduino.");
        publishProgress(temperatura);
        cierra();
        }
        return ok;
        }
        private void cierra() {
        close(mBufferedReader);
        close(lectura_entrada);
        close(flujo_entrada);
        close(mSocket);
        }
        private UUID uuid() {
        return UUID.fromString(UUID_SERIAL_PORT_PROFILE);
        }
        private void close(Closeable aConnectedObject) {
        if (aConnectedObject == null) return;
        try {
        aConnectedObject.close();
        } catch (IOException e) {
        e.printStackTrace();
        }
        aConnectedObject = null;
        }

       @Override
        protected void onProgressUpdate(Temperatura... values) {
        super.onProgressUpdate(values);
        callback.onTemperaturaActualizada(values[0]);
        }

        @Override
        protected void onCancelled() {
        callback.onCancelled();
        }
        }
```

CLASE Temperatura

```
   package org.example.junior.upnabluejrs;

/**
 * Created by JUNIORS on 06/09/2016.
 */
public class Temperatura {
    private String informacion = "";
    private String temperatura = "";
    private String nombreDispositivo = "";
    public String getInformacion() {
        return informacion;
    }
    public void setInformacion(String informacion) {
        this.informacion = informacion;
    }
    public String getTemperatura() {
        return temperatura;
    }
    public void setTemperatura(String temperatura) {
        this.temperatura = temperatura;
    }
    public String getNombreDispositivo() {
        return nombreDispositivo;
    }
    public void setNombreDispositivo(String nodo) {
        this.nombreDispositivo = nodo;
    }
}
```

CLASE AcercaDe

```
   package org.example.junior.upnabluejrs;

import android.app.Activity;
import android.os.Bundle;

/**
 * Created by JUNIORS on 07/09/2016.
 */
public class AcercaDe extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.acercade);
    }
}
```

FICHERO acercade.xml

```
   <?xml version="1.0" encoding="utf-8"?>
<TextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/TextView01"
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="La aplicación UpnaBueJR ha sido desarrollada por
Juniors Antonio Medina Landeon,está basada en la comunicación por
sockets Bluetooth entre Android y Arduino, y en el uso de hilos
Android-Java.">
</TextView>
```

FICHERO activity_main.xml

```
   <?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="org.example.junior.upnabluejrs.MainActivity"
    android:weightSum="10"
    android:orientation="vertical"

android:background="@color/abc_primary_text_disable_only_material_ligh
t">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/textView_Temperatura"
        android:textSize="20dp"
        android:layout_above="@+id/textView_Hora"
        android:layout_weight="1" />

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:id="@+id/textView_Hora"
        android:textSize="20dp" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Acerca de"
        android:id="@+id/button"

        android:onClick="lanzarAcercaDe" />
```

```
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="SALIR"
        android:onClick="salir"
        android:id="@+id/button2" />


</LinearLayout>
```

---------- MANIFEST.XML----------

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.juniors.controlcamclim" >
    <uses-permission android:name="android.permission.BLUETOOTH"/>
    <uses-permission
android:name="android.permission.BLUETOOTH_ADMIN"/>


    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:theme="@style/AppTheme.NoActionBar" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity   android:name=".AcercaDe"
            android:label="Acerca de ..."
            android:theme="@android:style/Theme.Dialog"/>
    </application>


</manifest>
```

ANEXO 2

Código de Arduino para la configuración del módulo bluetooth

```
#include <SoftwareSerial.h>

//Aquí conectamos los pins RXD,TDX del otro puerto serial del arduino

SoftwareSerial BT(10,11); //10 RX, 11 TX.



void setup() {



  pinMode (9,OUTPUT); //configuro el pin 9 como salida,correspondiente al PIN KEY del modulo bluetooth

  digitalWrite(9,HIGH); // convierto el pin 9 en estado alto, es decir se le entrega los 5v

  Serial.begin(38400); // la comunicacion serial entre el arduino y el ordenador a 9600 bps

  Serial.println("Por favor ingrese el comando AT: ");

  BT.begin(38400); // establezco una comunicación serial entre el modulo bluetooth y arduino a 38400 bps o baudios



}

void loop() {

   // Serial es como un canal, "un cable" y un write es enviar datos desde el arduino hacia el ordenador o hacia el modulo bluetooth
```

```
    /*-------SE ENVIA INFORMACION DESDE EL MODULO BLUETOOTH AL MONITOR SERIE------*/
     if(BT.available())
    {
      Serial.write(BT.read());
    }
    /*** AQUI ES AL REVÉS *///
    if(Serial.available())
    {
      BT.write(Serial.read());
    }
  }
```

Código del programa principal que adquiere los datos de temperatura y los envía por el puerto serie hacia el módulo BT

```
  /*--------- PROPIO DE LA COMUNICACIÓN SERIE ENTRE MODULO HC-05 Y ARDUINO--------*/
  #include <SoftwareSerial.h>
  //Aquí conectamos los pins RXD,TDX del otro puerto serial del arduino
  SoftwareSerial BT(10,11); //10 RX, 11 TX.

  const String nombre_del_nodo = "NODO_ARDUINO"; //Maximum 10 characters
```

```
/*------Código SHT15 sensirion----------*/

// comandos para el Sensirion

int Comando_temperatura  = B00000011;  // comando para leer temperatura

int Comando_humedad    = B00000101;  // comando para leer la humedad relativa


// especifico para el sensor Sensirion

int clockPin = 3;  // pin para el reloj

int dataPin  = 4;  // pin para datos

int ack;  // detectar posible ocurrencia de errores

int recojo_valT;

int recojo_valH;

float temp ;

float RH_lineal ;

//int numerobitsT = 14;     // especifico el numero de bits con los q voy a medir (es la precision), por defecto 14

//int numerobitsH = 12;     // especifico el numero de bits con los q voy a medir (es la precision), por defecto 12

float RHtrue ;       // aqui se pone la humedad relativa corregida por la temperatura

float Punto_Rocio  ;       // y aqui la temperatura de rocio


void setup() {

  Serial.begin(38400); // abre el puerto serie a 38400 baudios
```

```
    /*--------- PROPIO DE LA COMUNICACIÓN SERIE ENTRE MODULO HC-05 Y ARDUINO--------*/

    //BT.begin(38400); // establezco una comunicación serial entre el modulo bluetooth y arduino a 38400 bps o baudios

  }


  void loop() {

    // tiene que pasar 11 ms desde que se polariza el sensor para que éste alcance el estado de reposo

    delay(11);   // hay q esperar 11 ms desde el start up


    // leer la temperatura (por defecto en 14 bits) y convertirla a grados Celsius

    recojo_valT = leerValorTemperatura();              // lee el dato (integer), éste valor no es entendible

    temp = calcularTemperatura(recojo_valT);   // lo traduzco a grados Celsius, ésto sí que es un valor entendible ^^


    /*// leer la RH_lineal(por defecto en 12 bits) y convertirla a HR en tanto por ciento

    recojo_valH = leerValorHumedad();              // leo el dato (integer),éste valor no es entendible,

    RH_lineal = calcularHR(recojo_valH);  */       // lo traduzco a HR en tanto por ciento, ésto sí que es un valor entendible ^^


    /* // calculos adicionales
```

```
    //RHtrue = calcularRH_verdadera(temp, recojo_valT, RH_lineal);

    Punto_Rocio  = calcular_P_Rocio(RH_lineal, temp);

    //mostrar_pantalla(temp, RH_lineal, RHtrue, Punto_Rocio); */



    /* creo una variable de tipo String , donde contendrá el nombre del nodo y la temperatura */

    String a =nombre_del_nodo+","+ String(temp) + "\n"+ "\r";

    MonitorSerie(a);

    BT.begin(38400); // establezco una comunicación serial entre el modulo bluetooth y arduino a 38400 bps o baudios, es decir abre el canal serie

    BT.println(a); // Lo que hago aquí es enviar la cadena de caracteres al puerto serie virtual, que hemos creado gracias a la librería,que es donde estará conectado nuestro HC-05

    BT.end(); // cierro dicho puerto serie virtual,o mejor dicho cierra el canal serie



    // espero segundo y medio hasta la proxima medida (minimo 1000 milisec para no calentar el Sensirion, me lo indica el datasheet)

    delay(1500);

  }




  // funciones importantes del sensor
```

```
// leer el valor de temperatura que da el sensor (hace todo el proceso transparente)

int leerValorTemperatura() {

  int valT = 0;



  enviocomandoSHT(Comando_temperatura, dataPin, clockPin); // envio comando al sensor (petición de la medida de la temperatura)

  esperar_medidaSHT(dataPin);          // tiempo de adquisición del sensor variable ( tiempo en el que el sensor realiza la medición)

  valT = obtener_MSB_LSB(dataPin, clockPin);   // sólo me van a importar los dos primeros bytes de la trama que arroja el sensor (MSB y LSB)

  evitar_CRC_SHT(dataPin, clockPin);       // el 3º byte de la trama que retorna el sensor (correspondiente al CRC) no me importa

  return valT;

}



// leer el valor de Humedad relativa que da el sensor

int leerValorHumedad() {

  int valH = 0;



  enviocomandoSHT(Comando_humedad, dataPin, clockPin);   // envio comando al sensor (petición de la medida de la RH_linealedad)

  esperar_medidaSHT(dataPin);          // tiempo de adquisición del sensor variable

  valH = obtener_MSB_LSB(dataPin, clockPin);          // sólo me van a importar los dos primeros bytes de la trama que arroja el sensor ( MSB y LSB)
```

```
      evitar_CRC_SHT(dataPin, clockPin);                    // el 3º byte de la trama que retorna
el sensor (correspondiente al CRC) no me importa
      return valH;
    }

    // enviar comando al Sensirion
    void enviocomandoSHT(int comando, int dataPin, int clockPin) {
     int ack ;

     // empieza la transmision, o la secuencia "Transmission Start"
      pinMode(dataPin, OUTPUT);
      pinMode(clockPin, OUTPUT);
      digitalWrite(clockPin, LOW); // valor por defecto
      digitalWrite(dataPin, HIGH);// valor por defecto
      digitalWrite(clockPin, HIGH);
      digitalWrite(dataPin, LOW);
      digitalWrite(clockPin, LOW);
      digitalWrite(clockPin, HIGH);
      digitalWrite(dataPin, HIGH); // valor por defecto
      digitalWrite(clockPin, LOW); // valor por defecto
```

```
    // escribir el comando (el protocolo de transmisión de comandos está basado en un byte)
    shiftOut(dataPin, clockPin, MSBFIRST, comando);

    // verificar que obtenemos el ACKs adecuados por parte del SHT15(en respuesta al comando enviado por el micro)
    digitalWrite(clockPin, HIGH);
    pinMode(dataPin, INPUT);
    ack = digitalRead(dataPin);
    if (ack == LOW){
     Serial.println("el sensor ha recibido correctamente el comando para leer la temperatura en grados centigrados ");
    digitalWrite(clockPin, LOW);
    }
    else {
     Serial.println("el sensor no ha recibido el comando por parte del arduino");
     digitalWrite(clockPin, LOW);

    }
   }
   // esperar a la respuesta del Sensirion
   void esperar_medidaSHT(int dataPin) {
    int ack;
```

```
  pinMode(dataPin, INPUT);

  ack = digitalRead(dataPin);

  while (ack == HIGH){

    ack = digitalRead(dataPin);

    }

}

// obtener dato del Sensirion

int obtener_MSB_LSB(int dataPin, int clockPin) {

  int val;



  // obtener los MSB (bits mas significativos)

  pinMode(dataPin, INPUT);

  pinMode(clockPin, OUTPUT);

  val = shiftIn(dataPin, clockPin, MSBFIRST);

  val = val << 8; // los bits son desplazados 8 posiciones a la izquierda



  // envío el ack, después de recibir el byte MSB

  pinMode(dataPin, OUTPUT);

  digitalWrite(clockPin, HIGH);

  digitalWrite(dataPin, LOW);

  digitalWrite(clockPin, LOW);
```

```
  // obtener los LSB (bits menos significativos)
  pinMode(dataPin, INPUT);
  pinMode(clockPin, OUTPUT);
  val |= shiftIn(dataPin, clockPin, MSBFIRST); // equivalente a val = val |shiftIn(dataPin, clockPin, MSBFIRST), "|" es el operador OR bit a bit (bitwise or)
  return val;
 }

 // saltarse (no solicitar) la comprobacion CRC
 void evitar_CRC_SHT(int dataPin, int clockPin) {
  pinMode(dataPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  digitalWrite(clockPin, HIGH);
  digitalWrite(dataPin, HIGH);
  digitalWrite(clockPin, LOW);
 }

 // función que calcula la temperatura en grados Celsius
 float calcularTemperatura(int valorT) {
  float d1 = -40.1; //ya que en nuestro caso la VDD es de 5 voltios
  float d2 = 0.01; // ya que la resolucion de la medida de la temperatura por defecto es de 14 bits
```

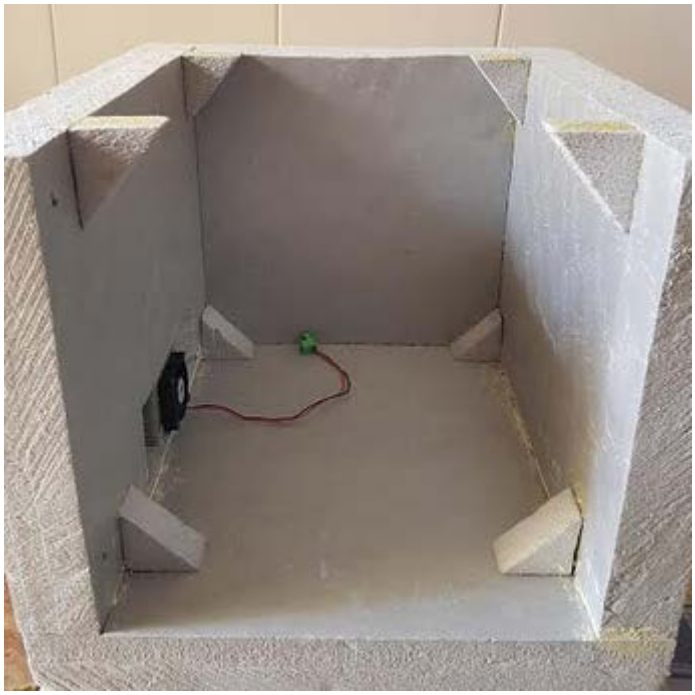
```
  float temperatura;



  temperatura = d1 + d2 * valorT;

  return temperatura;

}



// función que calcula la humedad relativa en porcentaje

float calcularHR(int valorH) {

  float c1 = -4.0;

  float c2= 0.0405;

  float  c3 = -0.0000028;

  float HR ;



  HR = c1 + c2 * valorH + c3 * sq(valorH);

  return HR;

}



// función que calcula el punto de rocio (DEW POINT)

float calcular_P_Rocio(float RH, float T) {

  float Tn_w = 243.12;  // por encima del agua ,de 0 a 50 grados Celsius

  float Tn_i = 272.62;  // por encima del hielo ,de -40 a 0 grados Celsius

  float m_w = 17.62;    // por encima del agua ,de 0 a 50 grados Celsius
```

```
    float m_i = 22.46;    // por encima del hielo, de -40 a 0 grados Celsius



    float Td ; // temperatura del punto de rocio

    float Tn ;

    float m ;



    if (T < 0.0) {   // los valores de las variables dependen de si la temperatura es mayor
o menor que cero

      Tn = Tn_w;

      m  = m_w;

    } else {

      Tn = Tn_i;

      m  = m_i;

    }

    Td = Tn * (log(RH/100) + ((m*T) / (Tn+T))) / (m - log(RH/100) - ((m*T) / (Tn*T)));

    return Td;  // devuelvo el valor del punto o temperatura de rocio

  }

  // calcular la Humedad relativa verdadera compensado por la temperatura

  float calcularRH_verdadera(float T, int valH, float HR) {

    float t1 = 0.01;

    float t2 = 0.00008; //resolucion de la medida por defecto de  la humeddad relativa
es 12 bits

    float HRtrue ;
```

```
  HRtrue = (T - 25.0) * (t1 + t2 * valH) + HR;
  return HRtrue;
}



// sacar por pantalla (se puede comentar al gusto, para no sacar tanta info)
void mostrar_pantalla(float T, float HR, float RH_verdadero, float Rocio) {
  Serial.print("    HR verdadera = ");
  Serial.print(RH_verdadero);
  Serial.print(" %,  T de rocio = ");
  Serial.print(Rocio);
  Serial.println(" C  ");
  Serial.print("Sensirion (T, HR) :  ");
  Serial.print(T);
  Serial.print(" C   ");
  Serial.print(HR);
  Serial.println(" %");
}
void MonitorSerie(String d) {
  Serial.print(d);
}
```

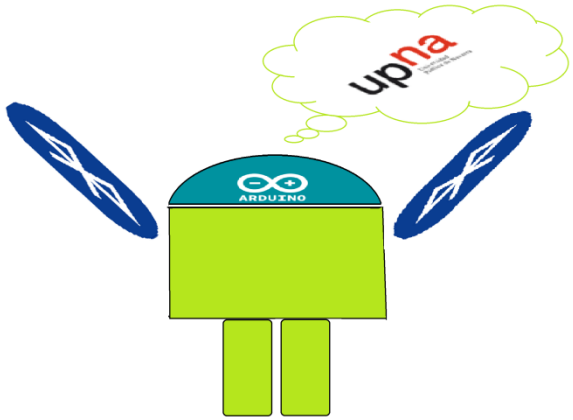
ANEXO 3

A continuación se muestran las imágenes de la cámara climática que construimos



Presentación PFC

Ingeniería de Telecomunicación



SISTEMA DE MONITORIZACIÓN INALÁMBRICO DE TEMPERATURA PARA DISPOSITIVOS ANDROID

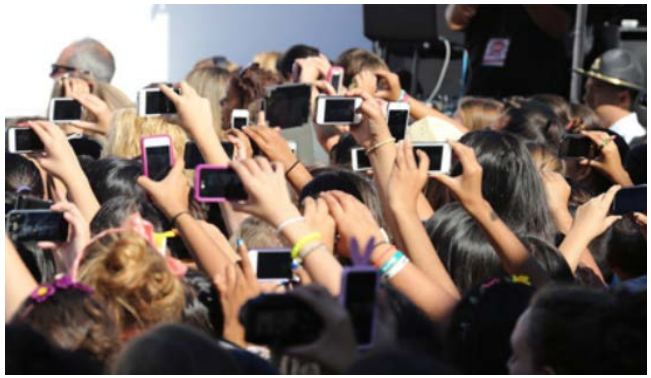
Juniors Antonio Medina Landeón
Tutores: Javier Goicoechea e Ignacio del Villar

1. INDICE

1. INTRODUCCIÓN Y OBJETIVOS

Contexto tecnológico actual:

- Dispositivos móviles: Todo el mundo tiene un ordenador en el bolsillo.



1. INTRODUCCIÓN Y OBJETIVOS

Contexto particular:

- En los laboratorios: cada equipo de medida tiene su ordenador al lado:
  - ☹☹ energía, sitio, calor…

Objetivos:

- Cámara climática de sobremesa (pequeña)
- Controlada por un microcontrolador
- Conexión a dispositivos móviles Android

1. INTRODUCCIÓN Y OBJETIVOS

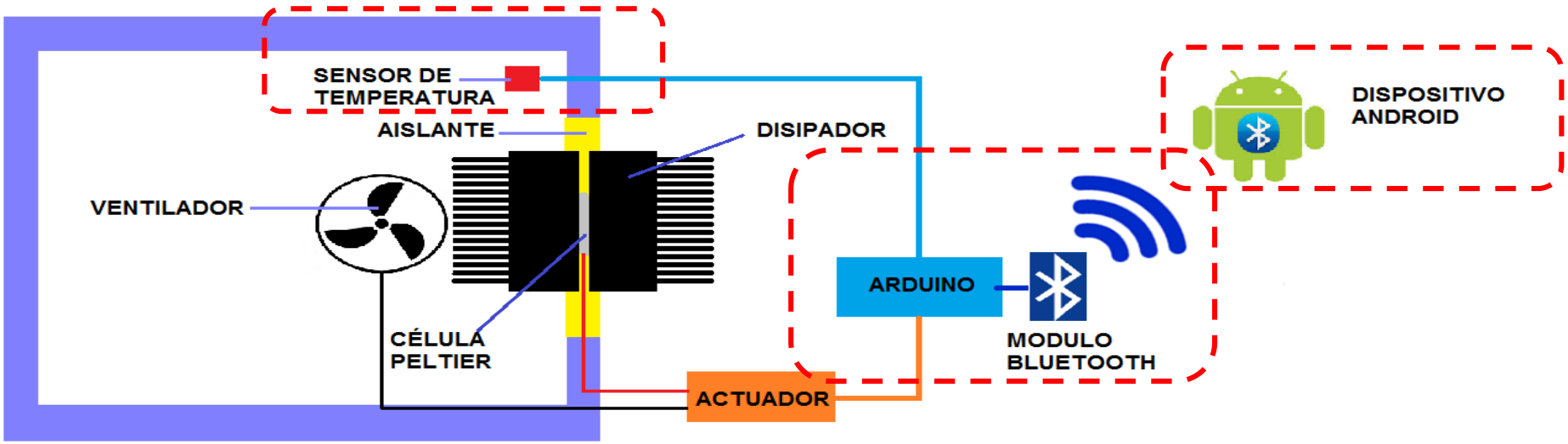
Objetivos:

- Cámara climática de sobremesa (pequeña)
- Controlada por un microcontrolador
- Conexión a dispositivos móviles Android

1. INTRODUCCIÓN Y OBJETIVOS

Cámara climática con conexión inalámbrica:

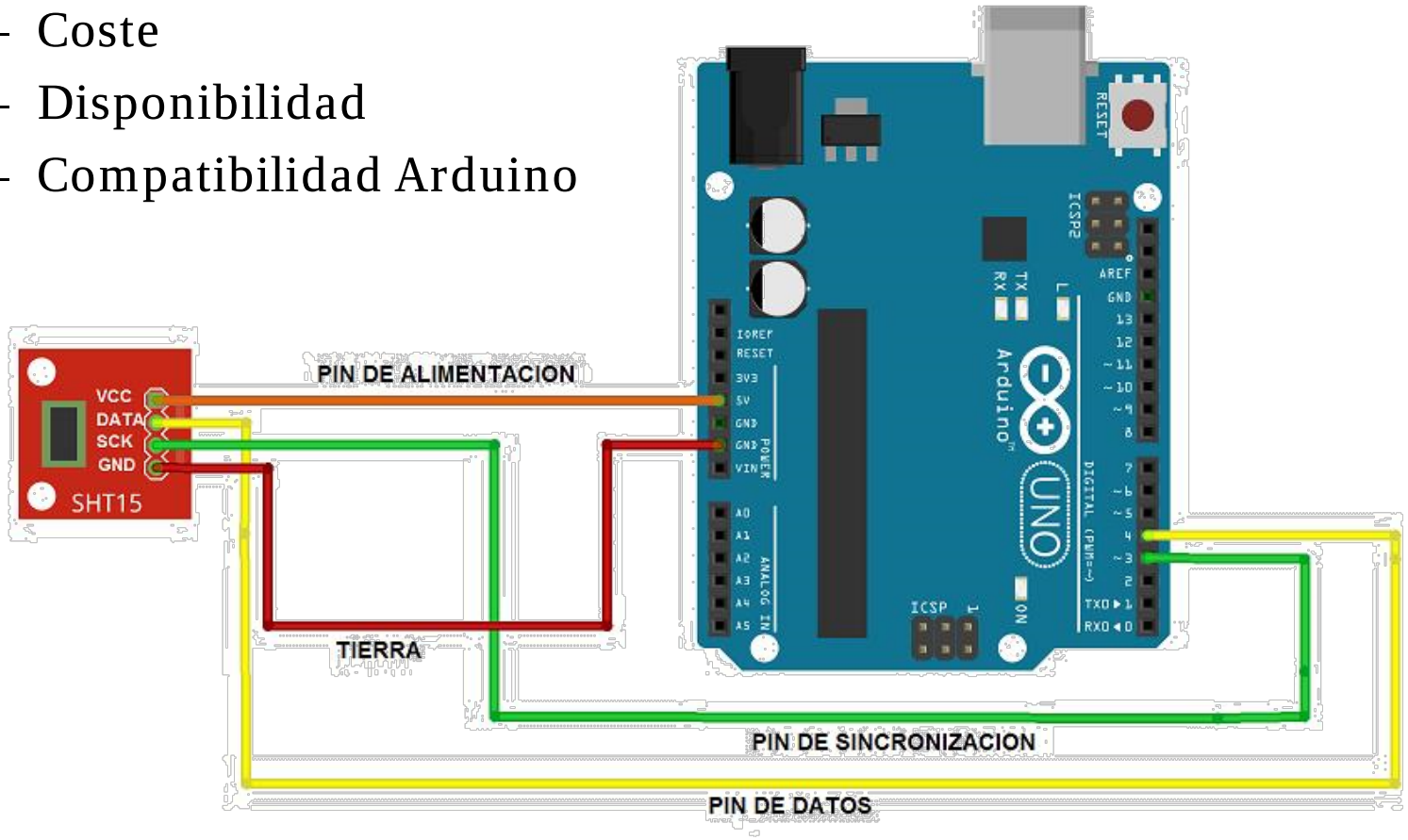
- Cámara de sobremesa
- Actuador Electrotérmico (Peltier)
- Microcontrolador Arduino MEGA
- Conexión inalámbrica Bluetooth



2. ADQUISICIÓN DE DATOS DE TEMPERATURA

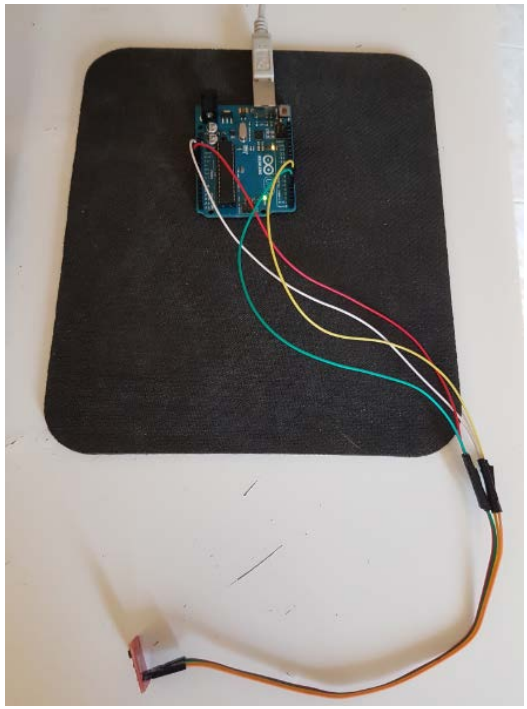
Sensor: SHT15

- ¿Porqué?
  - Coste
  - Disponibilidad
  - Compatibilidad Arduino

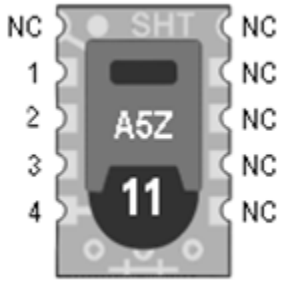


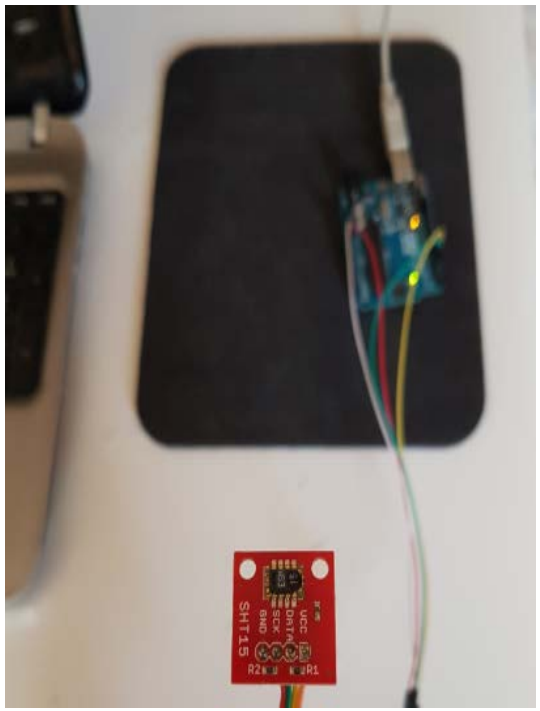
2. ADQUISICIÓN DE DATOS DE TEMPERATURA

Sensor: SHT15



Pin	Name	Comment
1	GND	Ground
2	DATA	Serial Data, bidirectional
3	SCK	Serial Clock, input only
4	VDD	Source Voltage
NC	NC	Must be left unconnected

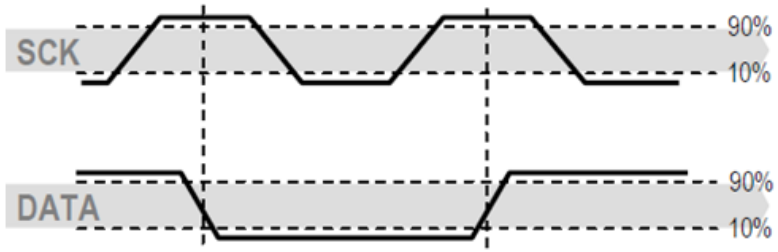




2. ADQUISICIÓN DE DATOS DE TEMPERATURA

Sensor: SHT15

- Principal inconveniente: Comunicación serie. Protocolo no estándar: necesidad de programar la comunicación

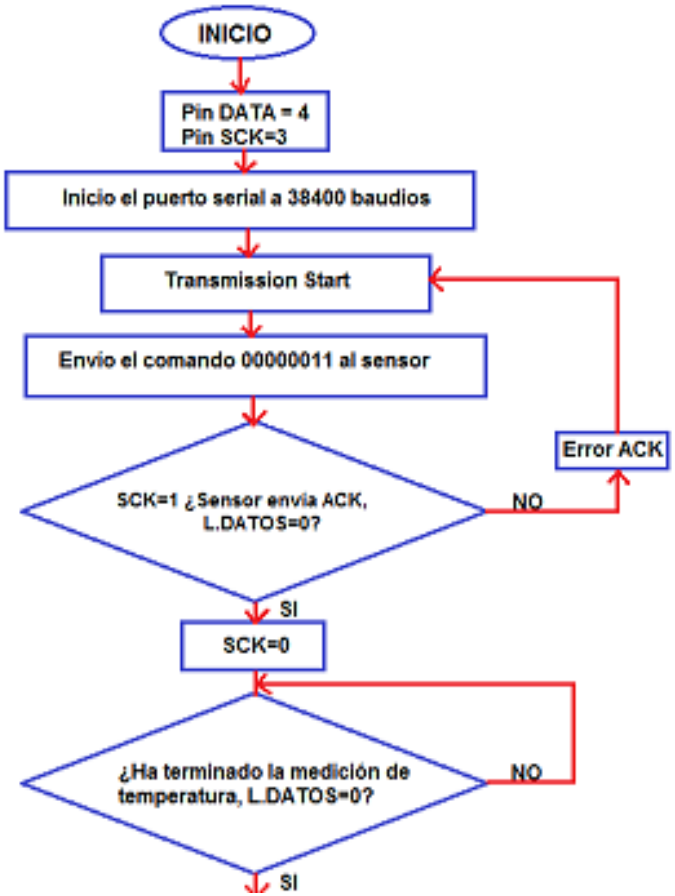


Command	Code
Reserved	0000x
Measure Temperature	**00011**
Measure Relative Humidity	**00101**
Read Status Register	00111
Write Status Register	00110
Reserved	0101x-1110x
Soft reset, resets the interface, clears the status register to default values. Wait minimum 11 ms before next command	**11110**

2. ADQUISICIÓN DE DATOS DE TEMPERATURA

Sensor: SHT15

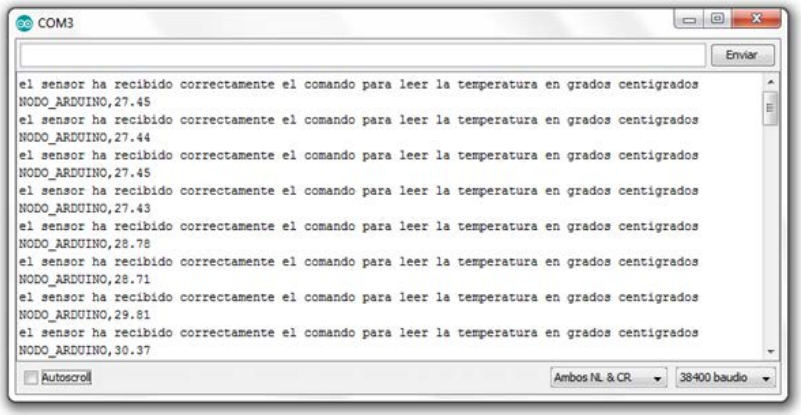
- Estructura del programa



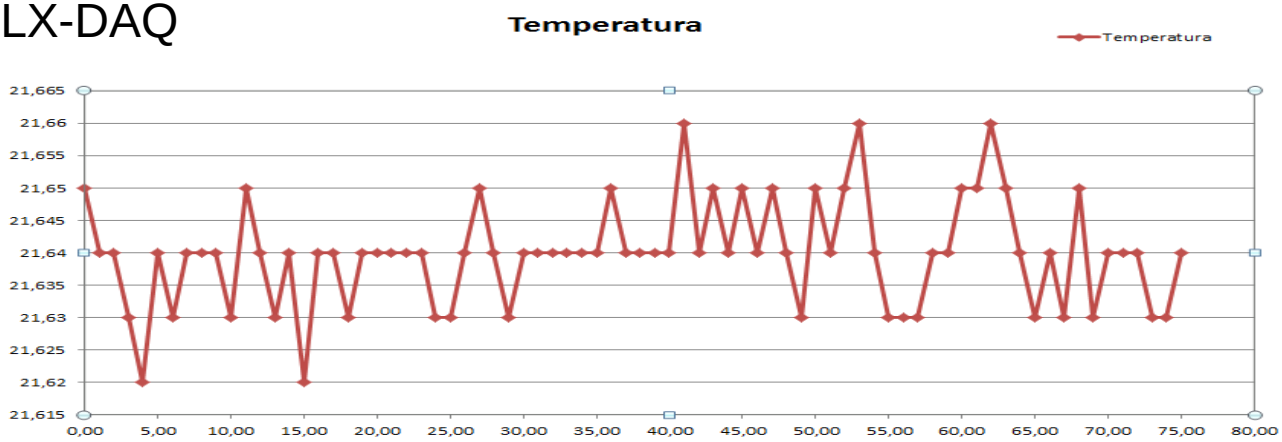
2. ADQUISICIÓN DE DATOS DE TEMPERATURA

Sensor: SHT15

- Demostración de funcionamiento



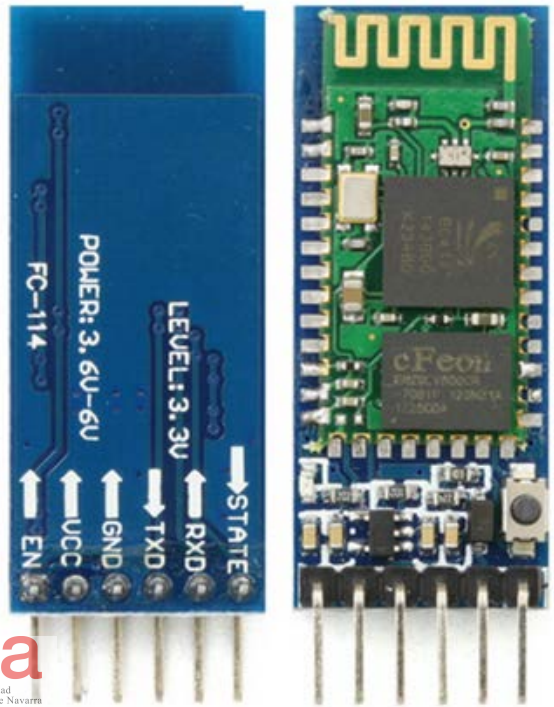
PLX-DAQ

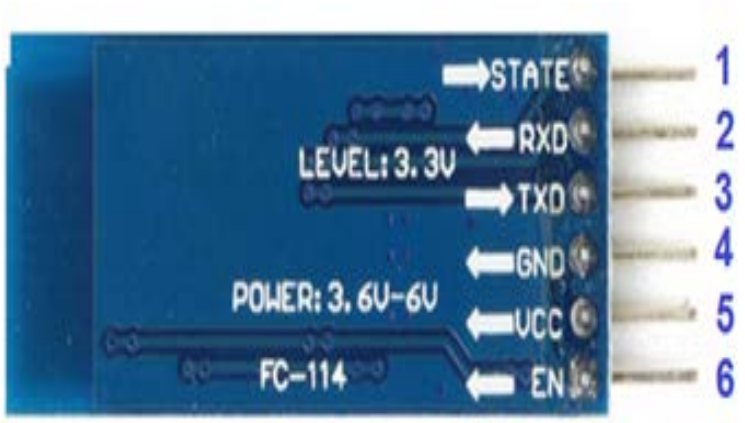


2. COMUNICACIÓN BLUETOOTH

Módulo Bluetooth:

- Módulo HC-05 FC-114
- Bajo coste, altas prestaciones
- Dos roles
- Seis pines, pero.....

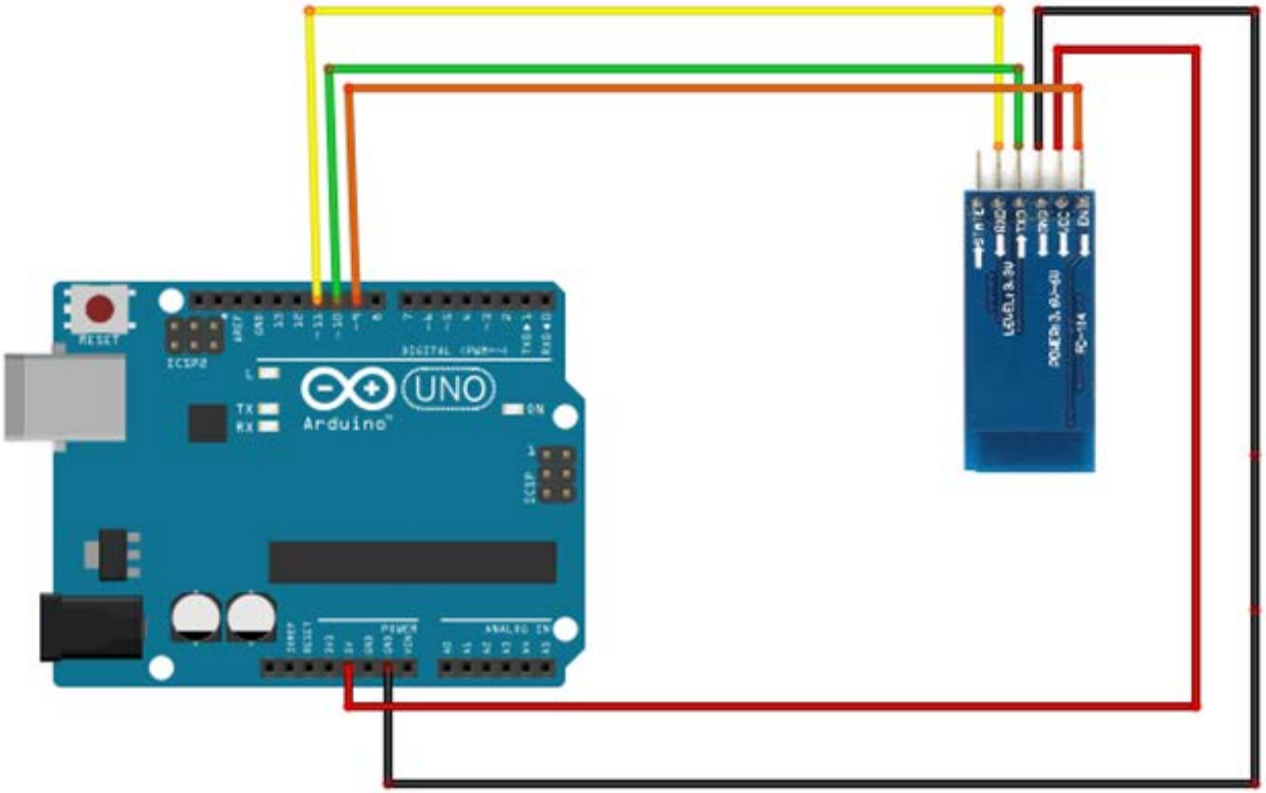




2. COMUNICACIÓN BLUETOOTH

Módulo Bluetooth:

- CONFIGURACION:



2. COMUNICACIÓN BLUETOOTH

Módulo Bluetooth:

- CONFIGURACION:

  - Cargar sketch en la placa arduino.

COMANDO_AT_JUNIOR Arduino 1.6.11

Archivo Editar Programa Herramientas Ayuda

COMANDO_AT_JUNIOR §

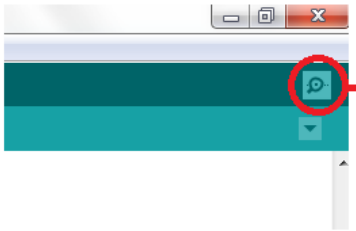
```
#include <SoftwareSerial.h>
//Aquí conectamos los pins RXD,TDX del otro puerto serial del arduino
SoftwareSerial BT(10,11); //10 RX, 11 TX.

void setup() {

  pinMode (9,OUTPUT); //configuro el pin 9 como salida,correspondiente al PIN KEY del modulo bluetooth
  digitalWrite(9,HIGH); // convierto el pin 9 en estado alto, es decir se le entrega los 5v
  Serial.begin(38400); // la comunicacion serial entre el arduino y el ordenador a 9600 bps
  Serial.println("Por favor ingrese el comando AT: ");
  BT.begin(38400); // establezco una comunicación serial entre el modulo bluetooth y arduino a 38400 bps o baudios

}
void loop() {
   // Serial es como un canal, "un cable" y un write es enviar datos desde el arduino hacia el ordenador o hacia el modulo bluetooth

  /*-------SE ENVIA INFORMACION DESDE EL MODULO BLUETOOTH AL MONITOR SERIE------*/
   if(BT.available())
  {
    Serial.write(BT.read());
  }
  /*** AQUI ES AL REVÉS *///
  if(Serial.available())
  {
     BT.write(Serial.read());
  }
}
```

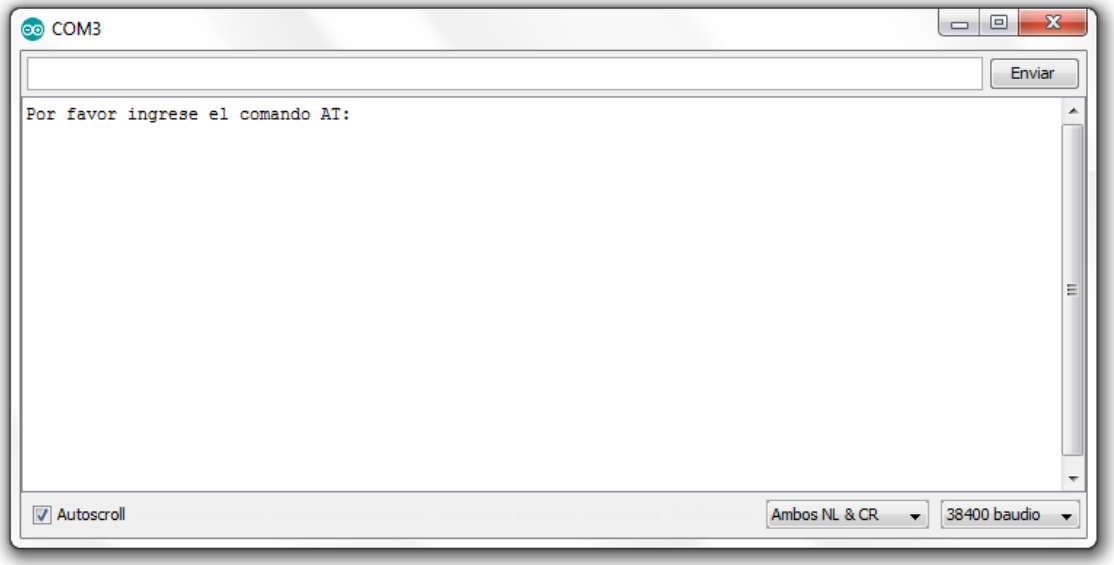
2. COMUNICACIÓN BLUETOOTH

Módulo Bluetooth:

- CONFIGURACION:

  - Abrimos el monitor serie del arduino.





2. COMUNICACIÓN BLUETOOTH

Módulo Bluetooth:

- CONFIGURACION:

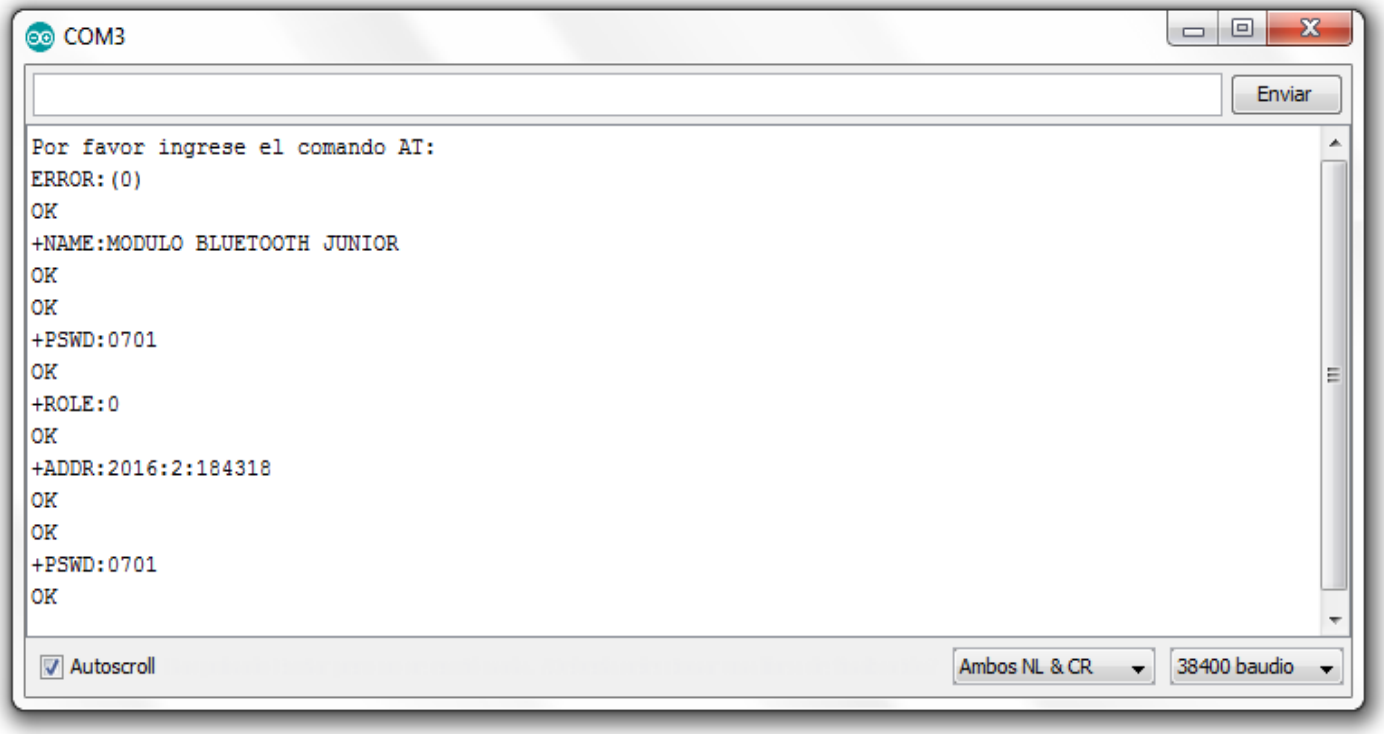
  - Introducción de los comandos AT.

```
AT      ----------> RETORNA LA RESPUESTA OK
AT+NAME? ----> MUESTRA EL NOMBRE ACTUAL
AT+NAME=MODULO BLUETOOTH JUNIOR ----------> CONFIGURO EL NOMBRE
AT+PSWD? -----> MUESTRA LA CONTRASEÑA
AT+PSWD=0701
AT+ROLE?-------> MUESTRA EL ROL DEL MODULO (MAESTRO/ESCLAVO)
0=ESCLAVO
1=MAESTRO
AT+ADDR?------> MUESTRA LA DIRECCION MAC DEL MODULO
```

2. COMUNICACIÓN BLUETOOTH

Módulo Bluetooth:

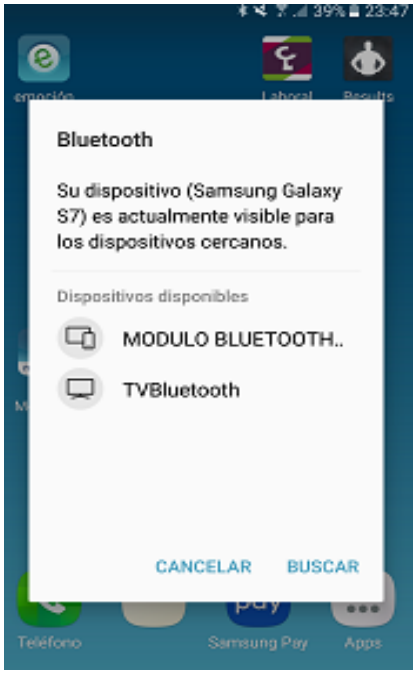
- CONFIGURACION:
  - Introducción de los comandos AT.

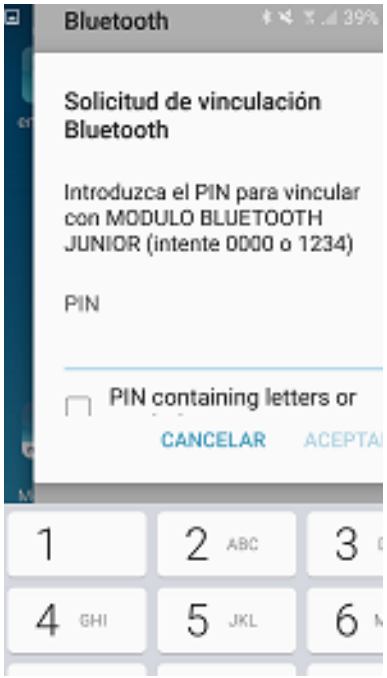


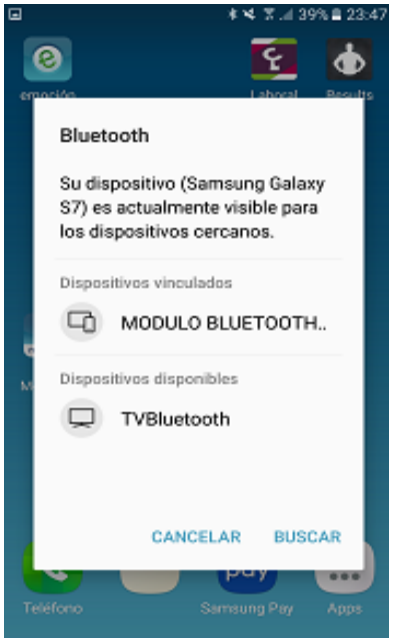
2. COMUNICACIÓN BLUETOOTH

Módulo Bluetooth-Dispositivo

- EMPAREJAMIENTO





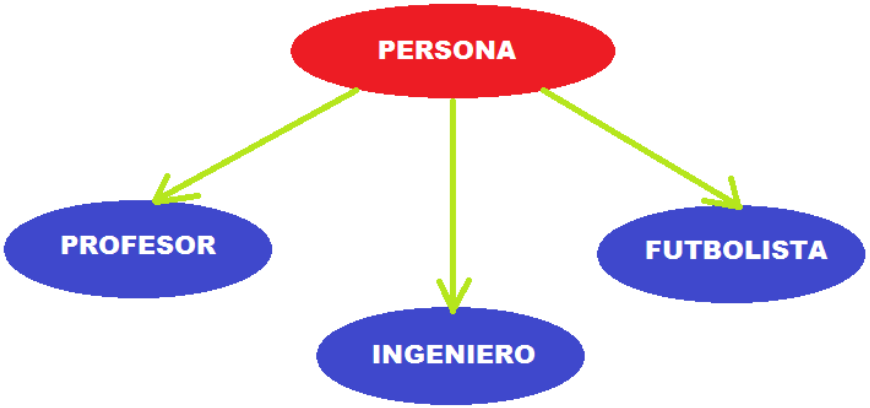


2. PROGRAMA ANDROID

- **Conceptos básicos Java**

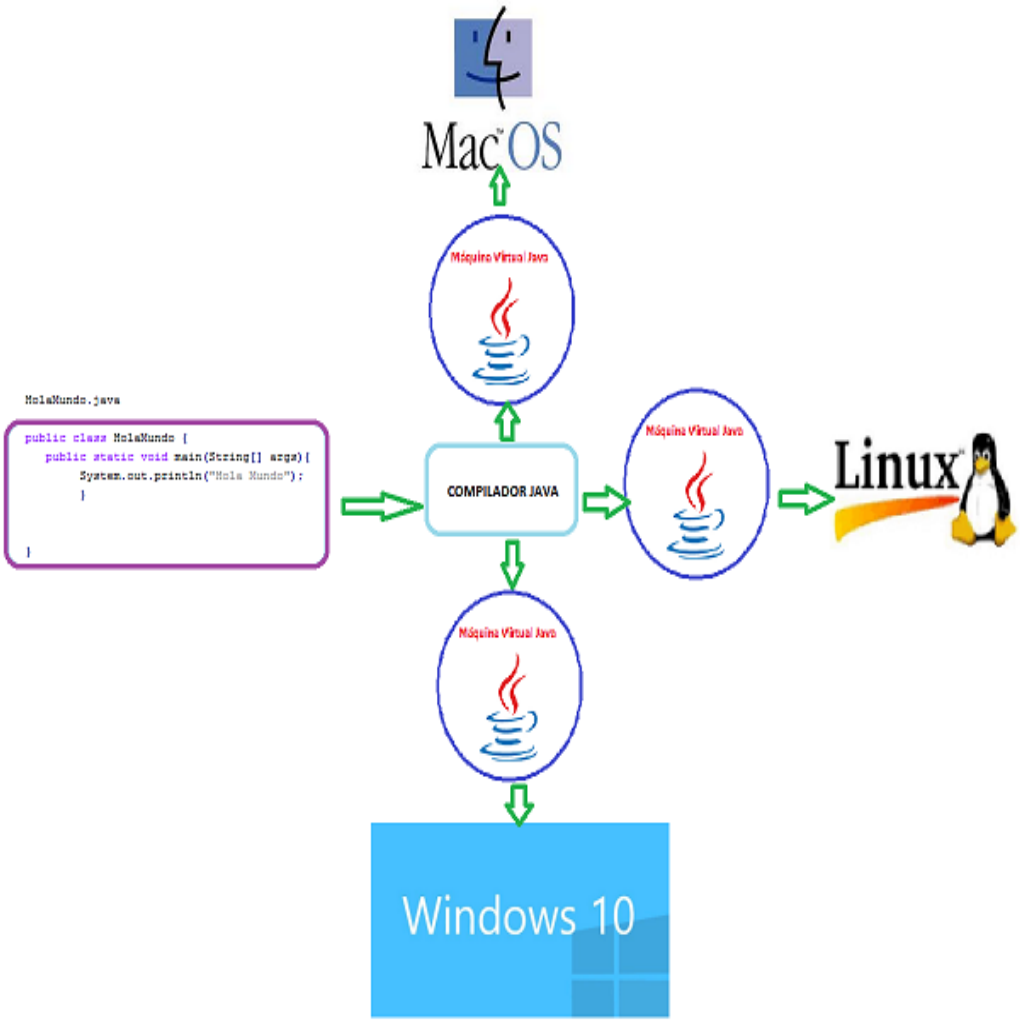
  Programar en Android es programar en Java.

  - Objeto.
  - Atributo
  - Método
  - Clases.
  - Interfaces
  - Herencia



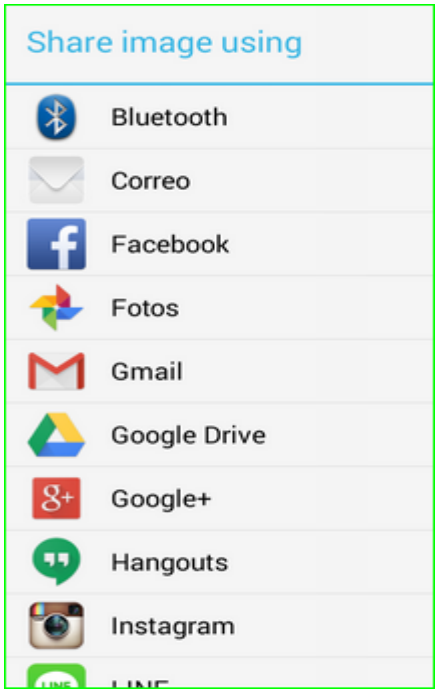
2. PROGRAMA ANDROID

- Algo muy importante de Java:



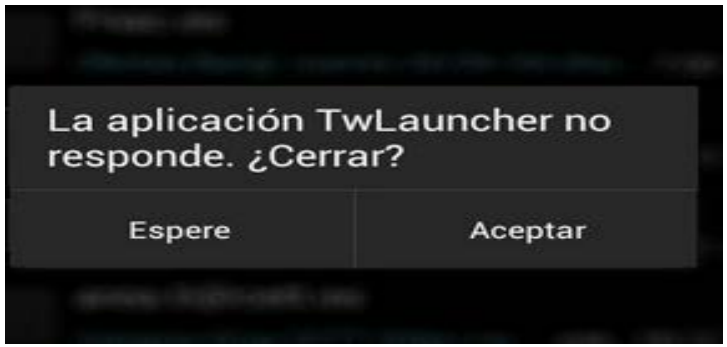
2. PROGRAMA ANDROID

- **Conceptos básicos Android**
  - Activity
  - Vista
  - Layout
  - Intent
  - Multithread

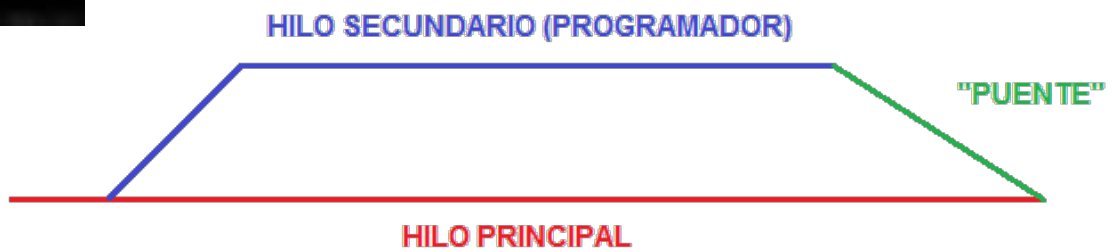


2. PROGRAMA ANDROID

- **Conceptos básicos Android**
  - Multithread
  - Asynctask

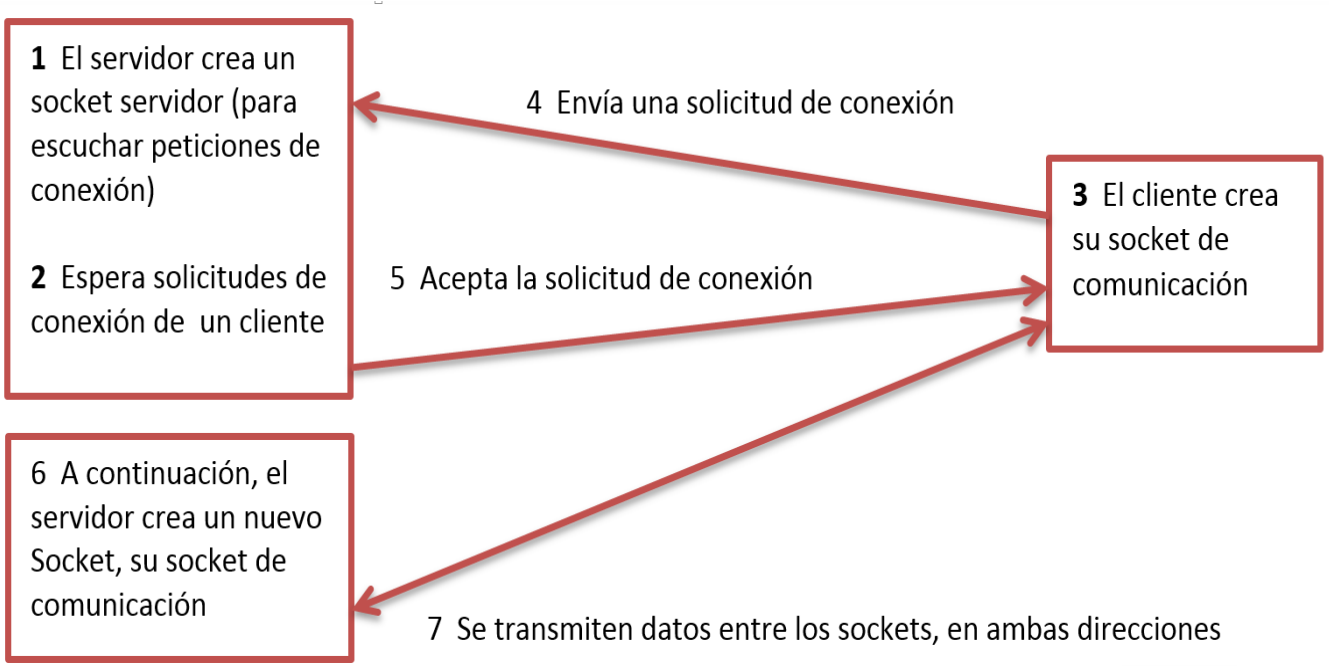



```
class MiTareaAsincrona extends AsyncTask
<Parametros,Progreso,Resultado>  {
…
}
```



2. PROGRAMA ANDROID

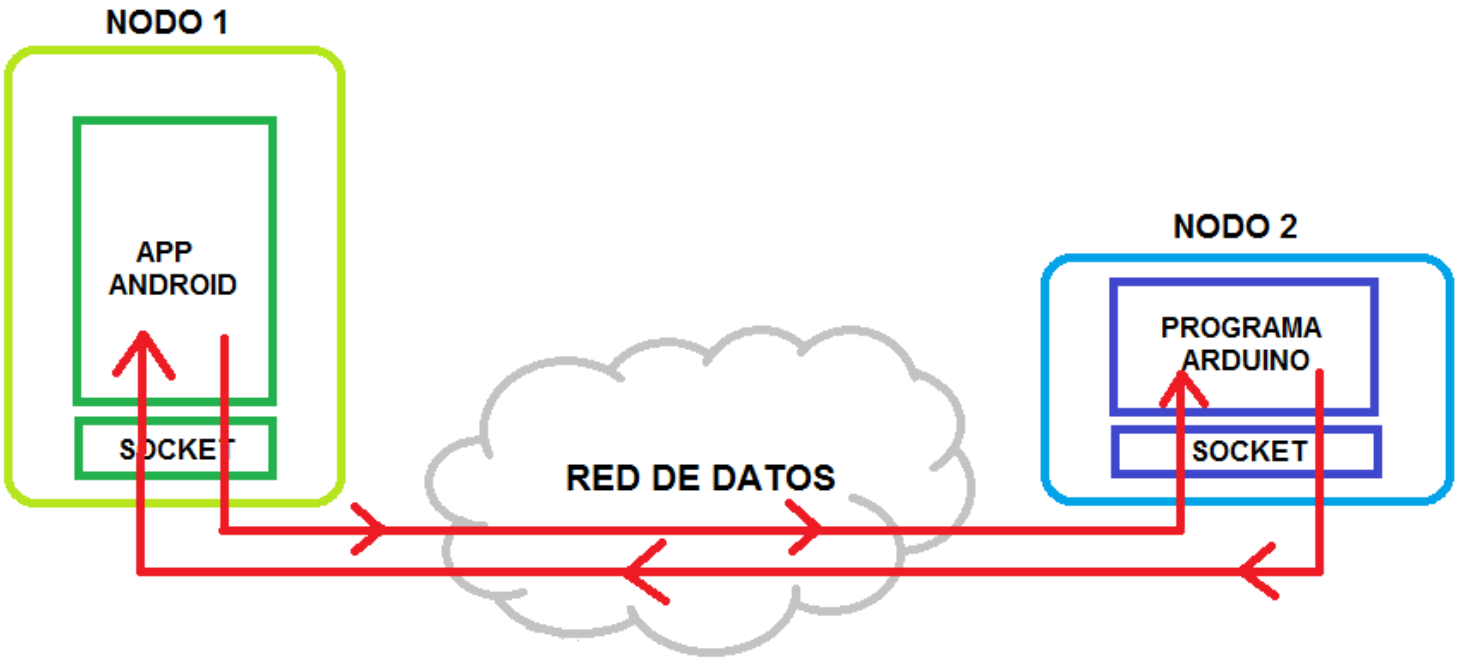
- **Comunicación Bluetooth en Android**



- Socket de servidor, UUID `-00001101-0000-1000-8000-00805F9B34FB`
- Sockets de comunicación

2. PROGRAMA ANDROID

- **Comunicación Bluetooth en Android**
- Socket de servidor, UUID
- Sockets de comunicación



2. PROGRAMA ANDROID

- **Estructura del programa**

- La aplicación va a constar de cuatro clases

2. PROGRAMA ANDROID

- **Estructura del programa**

- Main activity

MainActivity

-IU.
-Comprueba el adaptador Bluetooth.
-Comprueba si está activado.
-Obtener lista de emparejamiento.
-Crear tarea asincrona

Metodos utilizados:
-getDefaultAdapter().
-getBondedDevice()
Clases utilizadas:
-BluetoothAdapter
-BluetoothDevice

2. PROGRAMA ANDROID

- **Estructura del programa**

- MiTareaAsync

MiTareaAsync

-Clase descendiente de Asynctask.
-Se declara UUID del Arduino.
-Recibe parámetro.
-Crea socket de cliente.
-Tarea costosa.

Metodos utilizados:
-
createRfcommSocketToServiceRecord(UUID)
-connect()
Clases utilizadas
-BluetoothSocket
-InputStream

2. PROGRAMA ANDROID

- **Estructura del programa**

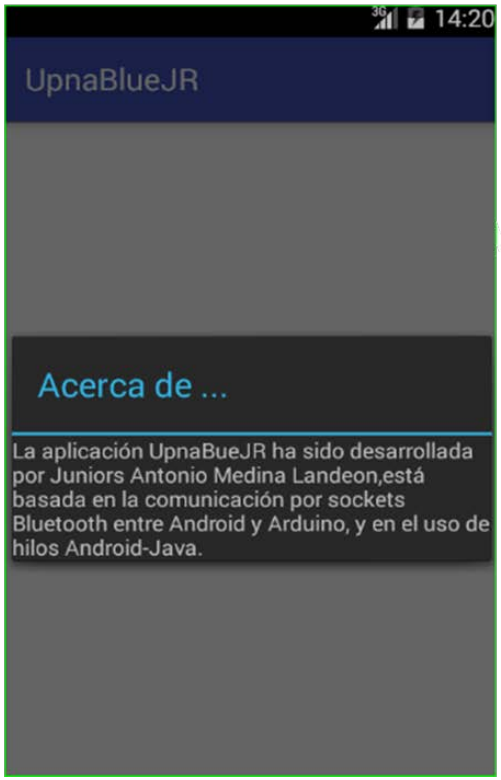
- Temperatura

Temperatura

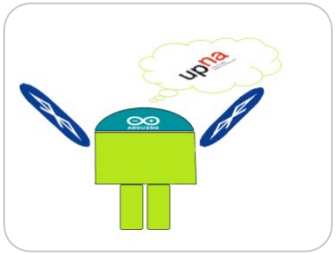
- CLASE AUXILIAR

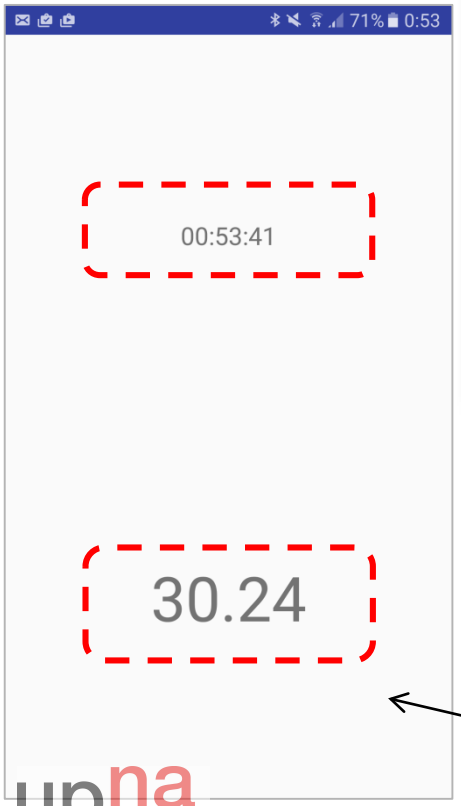
2. PROGRAMA ANDROID

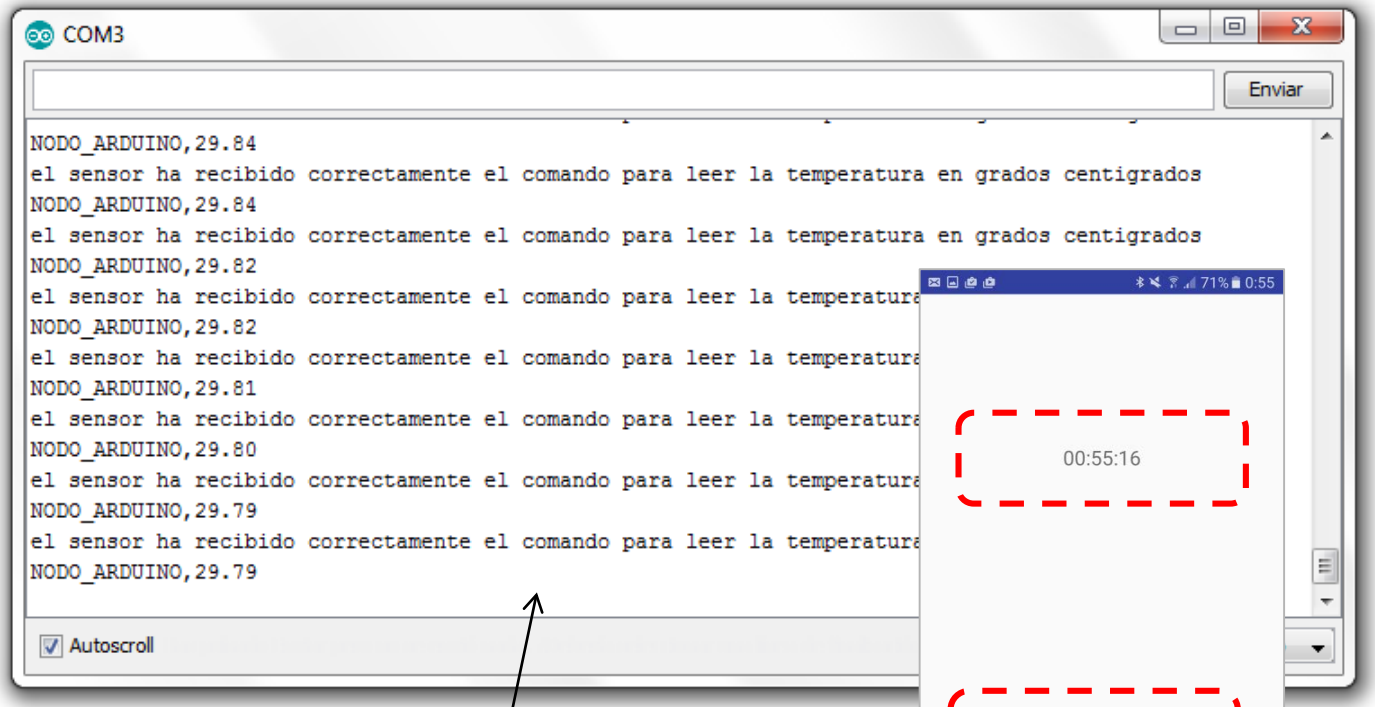
- **Estructura del programa**
  - AcercaDe



2. DEMOSTRACIÓN DE FUNCIONAMIENTO







Captura del monitor serie

Captura de pantalla del móvil

2. CONCLUSIONES Y LINEAS FUTURAS

- **Conclusiones**
  - Se ha conseguido establecer una comunicación Bluetooth entre el microcontrolador encargado del gobierno de la cámara climática y un dispositivo móvil.
  - Se ha programado el microcontrolador Arduino para la interrogación del módulo sensor SHT15.
  - Se ha seleccionado y configurado un módulo de transmisión Bluetooth para la comunicación del Arduino con el dispositivo móvil.
  - Se ha desarrollado una aplicación Android que es capaz de conectarse con el microcontrolador Arduino y recibir los datos de temperatura que éste envía en tiempo real.
  - Problemas.
    - la mayor parte de la bibliografía: programación de sistemas de comunicaciones bluetooth entre dos dispositivos Android.
    - Sin embargo no es trivial la adecuación de este sistema de comunicación a una configuración Android-microcontrolador.

2. CONCLUSIONES Y LINEAS FUTURAS

- **Conclusiones**
  - Finalización de la cámara climática completando la parte de electrónica de potencia y del sistema de regulación automático de la temperatura.
  - Mejorar la interfaz gráfica
  - Establecimiento de una comunicación bidireccional. Esto permitiría que desde el dispositivo Android se pudieran enviar comandos al microcontrolador y así poder tener un control global.

Gracias por sus preguntas

Gracias por su atención